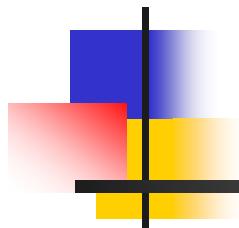


言語・オートマトン イントロダクション

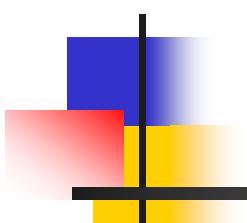


山本章博

情報学研究科 知能情報学専攻

2017.10.2

連絡先: akihiro@i.kyoto-u.ac.jp

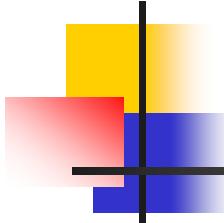


2016年度
言語・オートマトン

山本章博

連絡先: akihiro@i.kyoto-u.ac.jp

075-753-5995

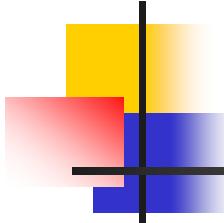


講義計画

内容:本講義では、形式言語理論と計算理論の初步を扱う。

形式言語理論は、プログラミング言語を設計する基盤になっているほか、マークアップ言語や自然言語処理、生命情報学など現代の情報学の様々な分野で応用されている。また、計算理論は計算機による処理の特徴と限界を明らかにする理論である。

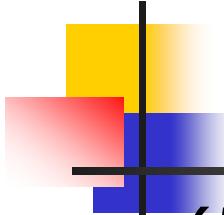
本講義では、有限オートマトンについて述べ、さらに文脈自由言語やチューリング機械、帰納的関数などについて講述する。また、これらの応用についても適宜言及する。



教科書・参考書

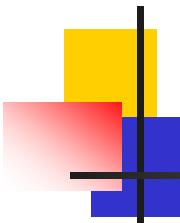
参考書

- Hopcroft, Motwani, Ullman: Introduction to Automata Theory, Languages, and Computation: 3rd edition, Pearson, 2007
- 富田・横森:オートマトン・言語理論:第2版,森北出版 2013
- 岡留:オートマトンと形式言語入門, 森北出版, 2009
- 有川(監修)西野・石坂(著) 形式言語の理論, 丸善出版, 1999



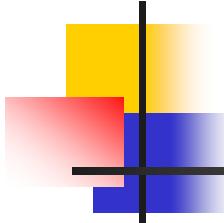
講義資料

- (ほぼ)毎回、講義資料をKULASISを用いて電子的に配布して使用する。
 - KULASISが利用できない場合は山本(章)のHPで配布



評価方法

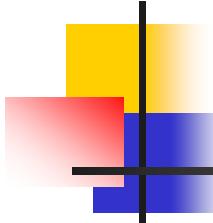
- 小テスト(随時)+定期試験
 - 小テストを行わない週には、講義の理解度を確認する演習問題を出題(評価に含めない)



形式言語理論

言語を文字列・単語列の集合と抽象化した上で

- 文を生成(構成・定義)するための理論
 - 正しい文字列・単語列を生成(定義)する
- 文を受理するための理論
 - 送られてきた文字列・単語列が正しいかどうかを判定する
- 句構造文法と受理機械



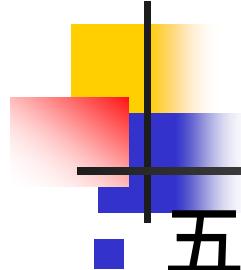
文法の型

- 文脈自由文法(context free) 文法中の全ての生成規則 $\alpha \rightarrow \beta$ について、 α が非終端記号一つであるもの
 - 生成規則に制限を加えることで、様々な文法の型が導入できる。

0型	句構造文法	Turing Machine
1型	文脈依存文法	線形有界オートマトン
2型	文脈自由文法	非決定性プッシュダウン・オートマトン
3型	正則文法	有限状態オートマトン



計算機科学と言語



情報とメディア

- 五感：発話, 手話, 表情, ...



- 媒体(media)：

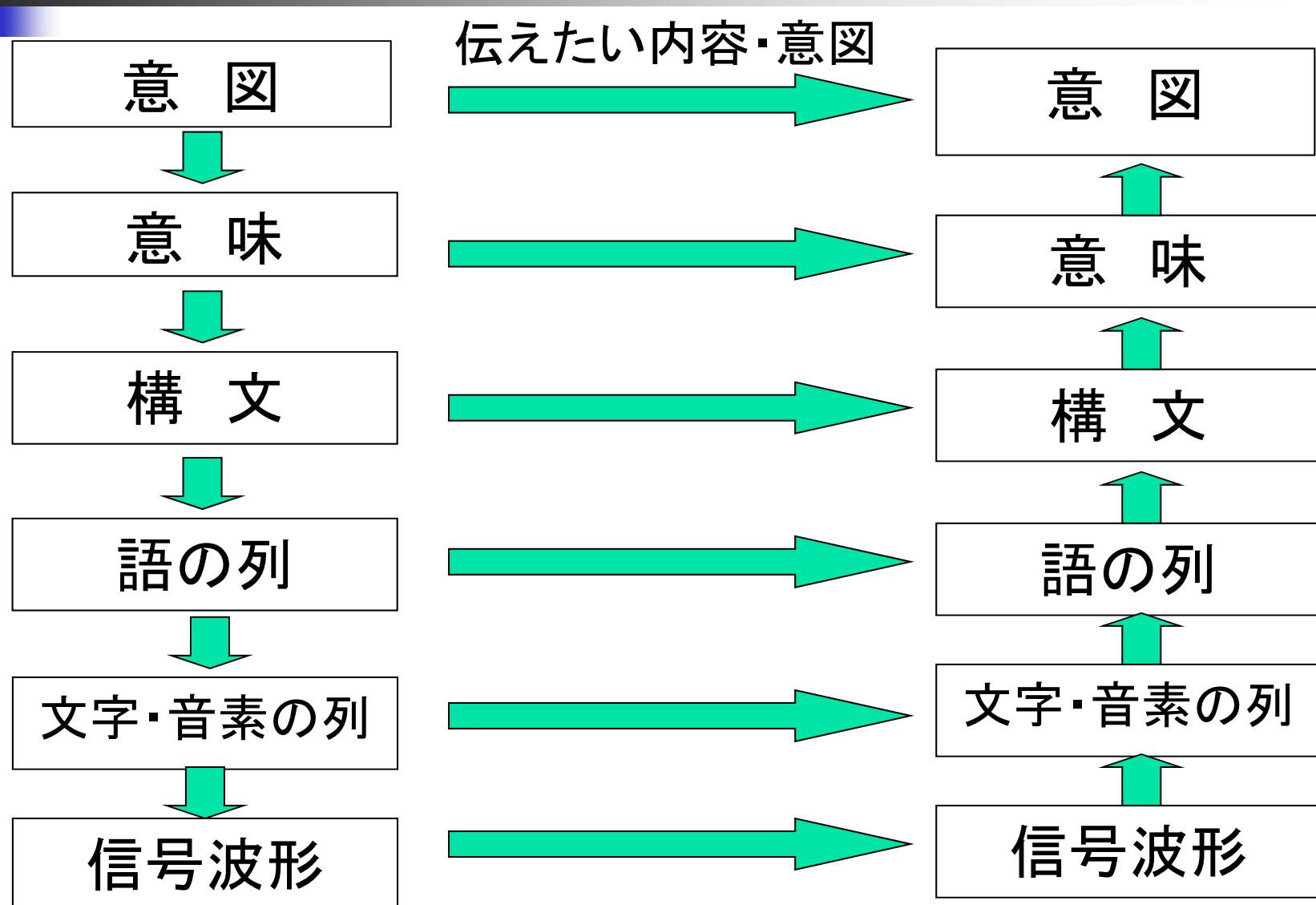
手紙, 書籍, 新聞, 電話, 入試問題, ...

Telephone, Telefacsimile, Television, ...

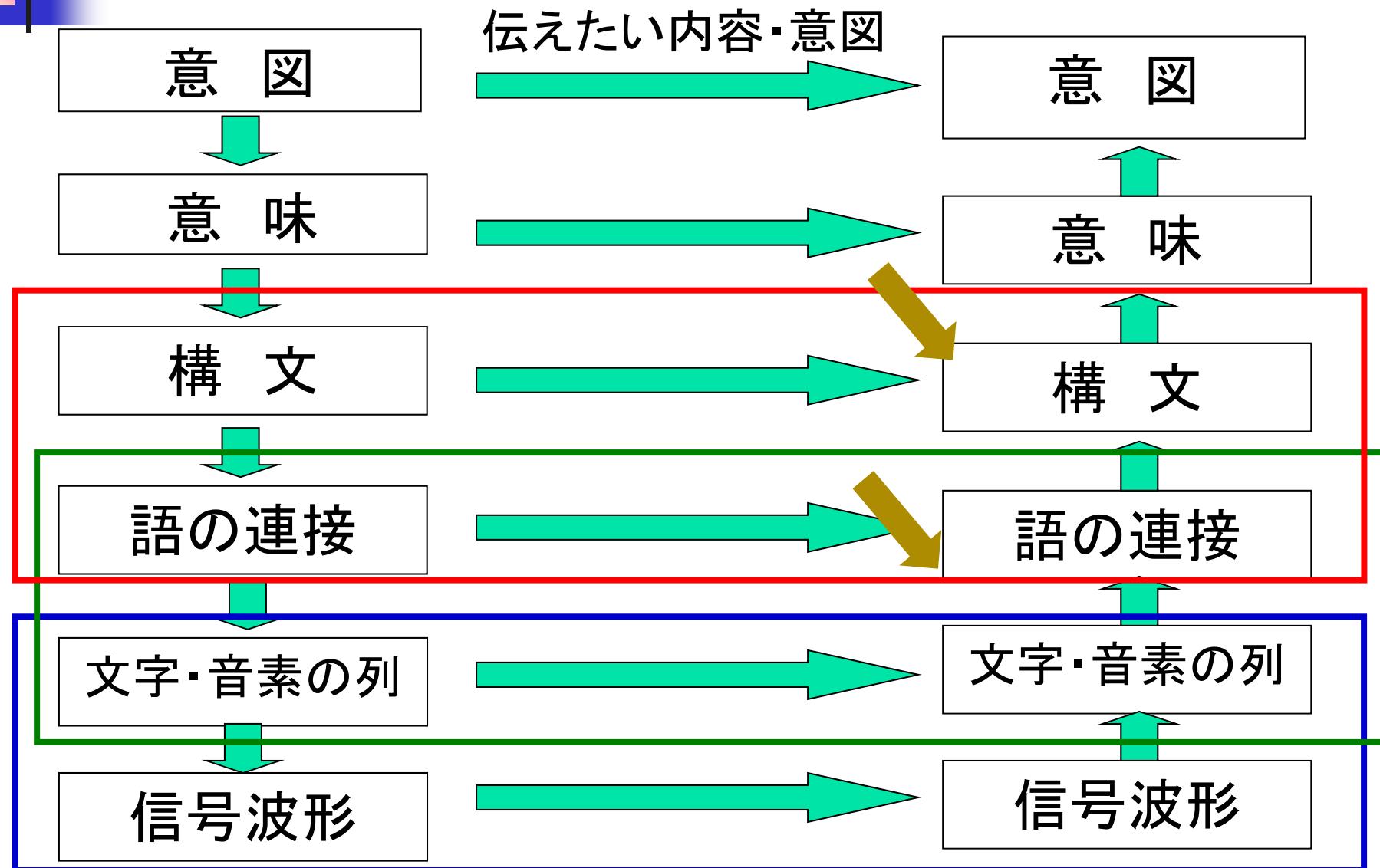
Tele : (Gk) far off



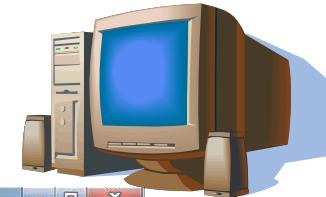
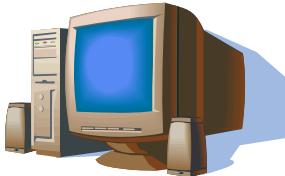
言語の構成要素と伝達



言語の構成要素と伝達



コンピュータ・ネットワーク



ホーム - 京都大学

www.kyoto-u.ac.jp/ja

よく見るページ Firefox を使いこなそう Cooperative Linux(... KNOPPIX Japanese ... UNIX USER2004年9... Yahoo! JAPANのビ... ビギナーズ Portable ...

Kyoto University

京都大学

日本語 ENGLISH CHINESE(簡体) CHINESE(繁体) KOREAN

受験生の方 一般の方 企業の方 OB・OGの方 在学生の方 教職員

イベントカレンダー 刊行物・資料請求 お問合せ アクセス・マップ サイトマップ リンク

現在の場所: ホーム 最新のニュース写真

総長VOICE
Office of the president

京都大学
オープンキャンパス
2014/8/7-8

KYOTO UNIVERSITY

消防車

ダイニン四迷路
キオシン四迷路
ダイニンチーム
キオシンチーム

東日本大震災へ

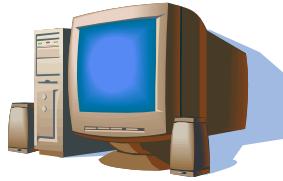
大学院生のため
教育実践講座2

● ニュース → 大学の動き → 研究成果 → 入試関連情報

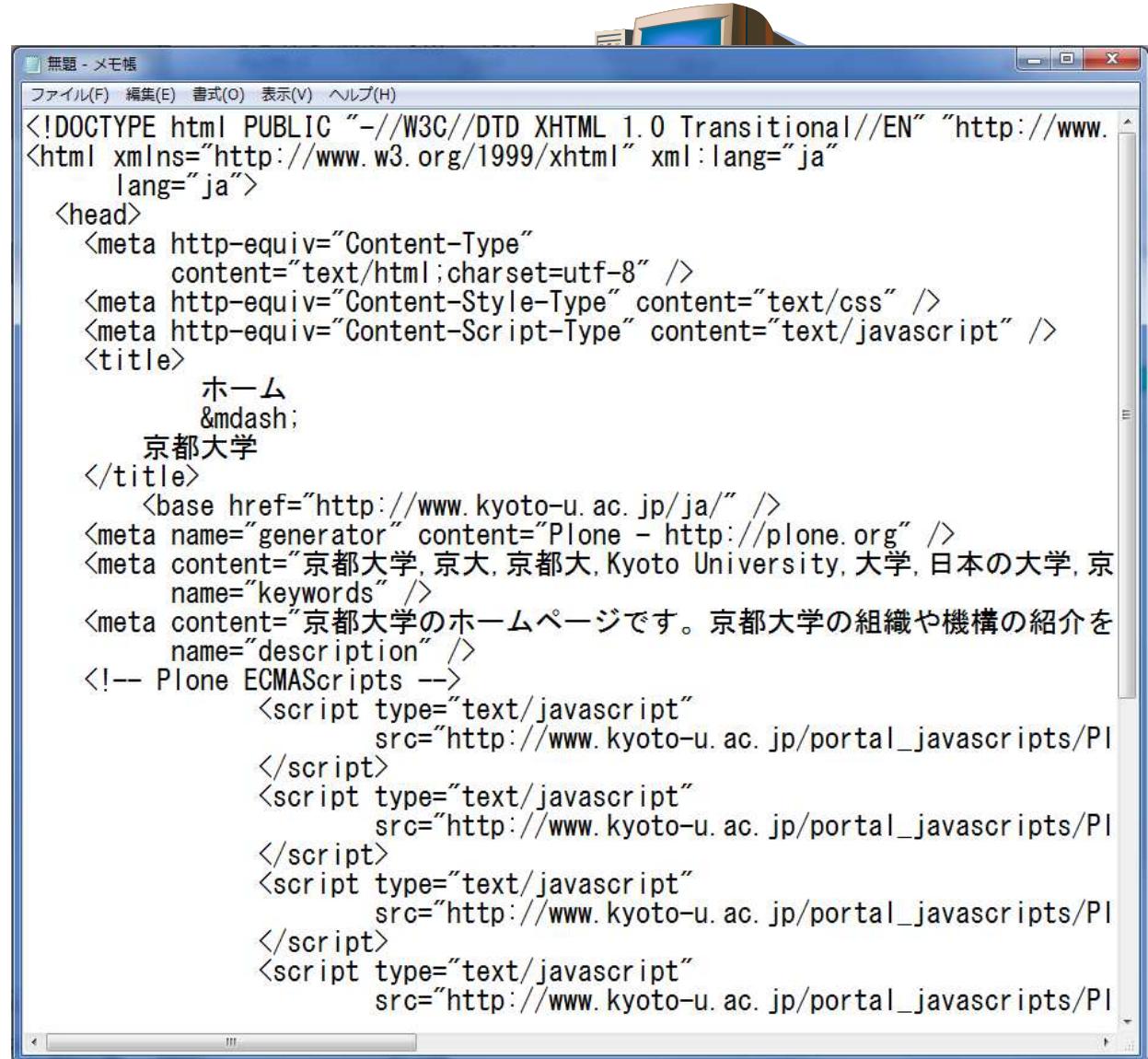
* 今吉格 白眉センター／ウイルス研究所特定准教授が、ドイツ・イノベーション・アワード「グットフリード・ワグネル賞 2014」（最優秀賞）を受賞しました。（2014年6月）



コンピュータ・ネットワーク

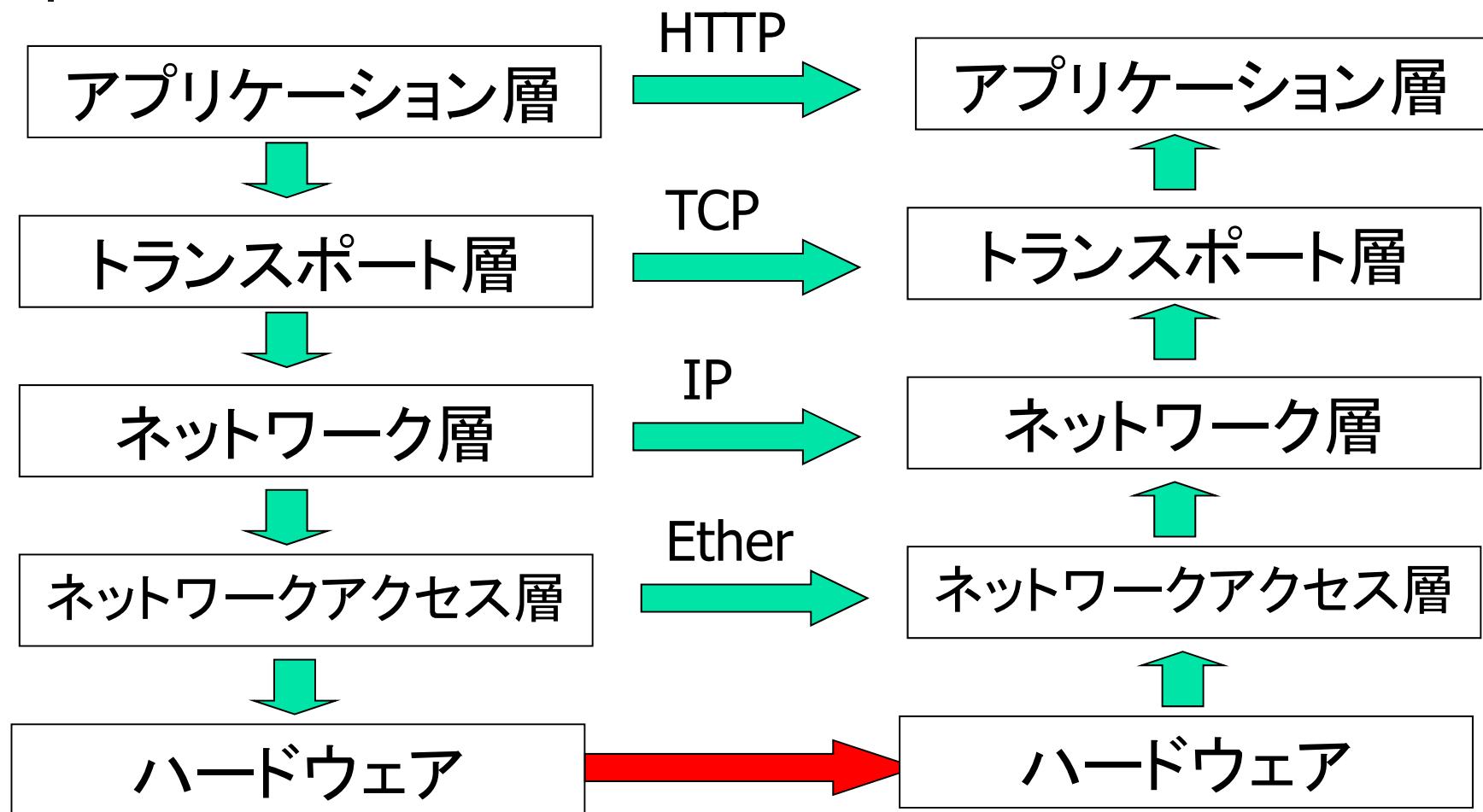


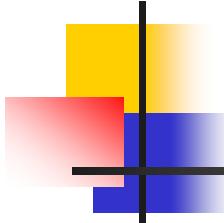
- HTML
- 文字列
- ビット列
(0, 1の列)
- 電気(光)信号



```
無題 - メモ帳
ファイル(F) 編集(E) 書式(O) 表示(V) ヘルプ(H)
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/1999/xhtml" xml:lang="ja" lang="ja">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <meta http-equiv="Content-Style-Type" content="text/css" />
  <meta http-equiv="Content-Script-Type" content="text/javascript" />
  <title>
    ホーム
    &mdash;
    京都大学
  </title>
  <base href="http://www.kyoto-u.ac.jp/ja/" />
  <meta name="generator" content="Plone - http://plone.org" />
  <meta content="京都大学, 京大, 京都大, Kyoto University, 大学, 日本の大学, 京" name="keywords" />
  <meta content="京都大学のホームページです。京都大学の組織や機構の紹介を" name="description" />
  <!-- Plone ECMA Scripts -->
  <script type="text/javascript" src="http://www.kyoto-u.ac.jp/portal_javascripts/Plone/ecmaScripts.js" />
  <script type="text/javascript" src="http://www.kyoto-u.ac.jp/portal_javascripts/Plone/ecmaScripts.js" />
  <script type="text/javascript" src="http://www.kyoto-u.ac.jp/portal_javascripts/Plone/ecmaScripts.js" />
  <script type="text/javascript" src="http://www.kyoto-u.ac.jp/portal_javascripts/Plone/ecmaScripts.js" />
```

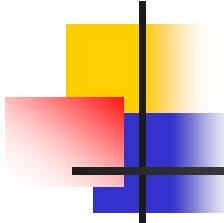
TCP/IP プロトコルの階層





自然言語の“文”的例

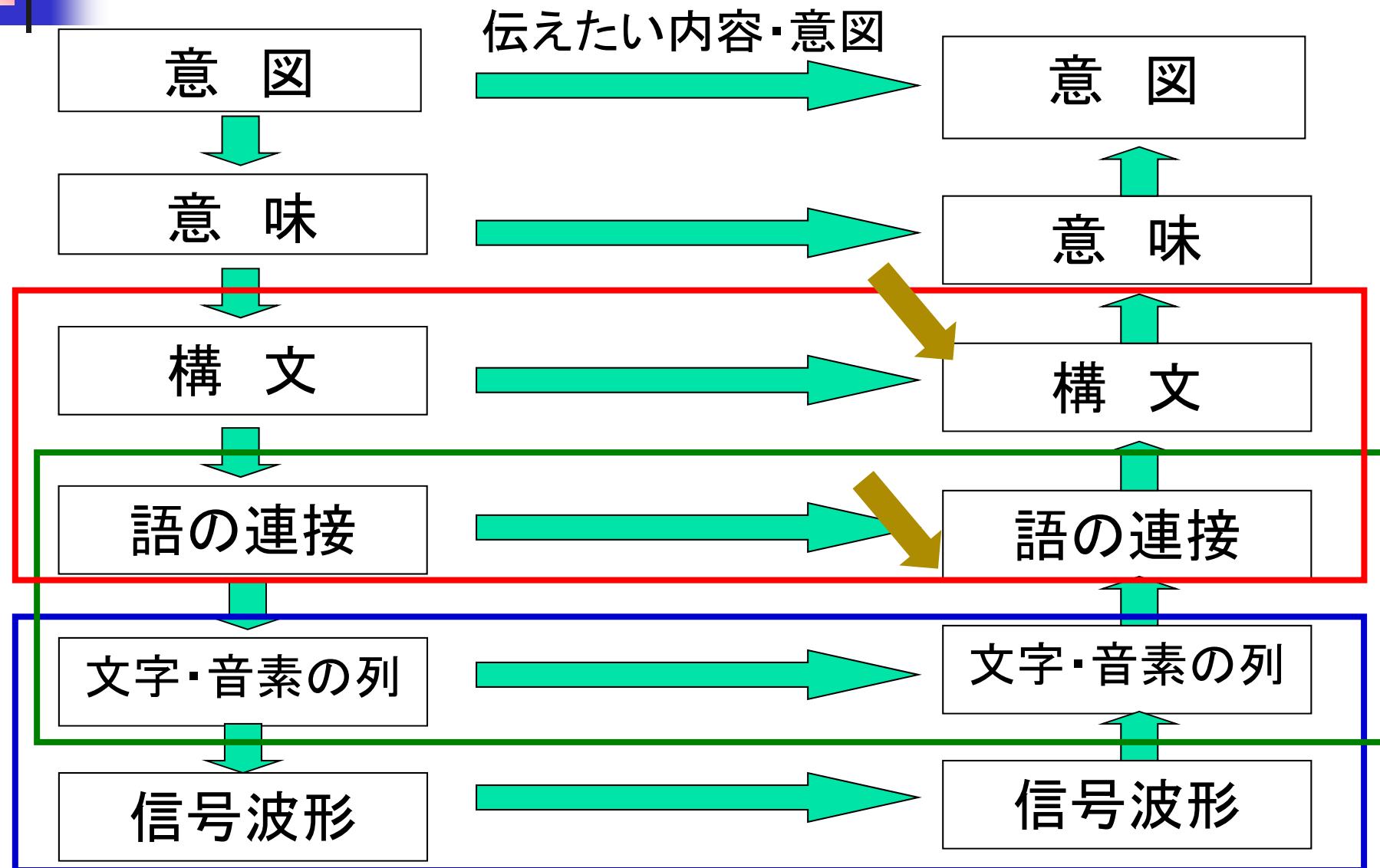
- 私は京都が好きです.
 - 京都は私のお気に入りです.
 - 京都って、いいよね.
-
- I like Kyoto.
 - Kyoto is one of my favorite cities.
 - Kyoto, my favorite!



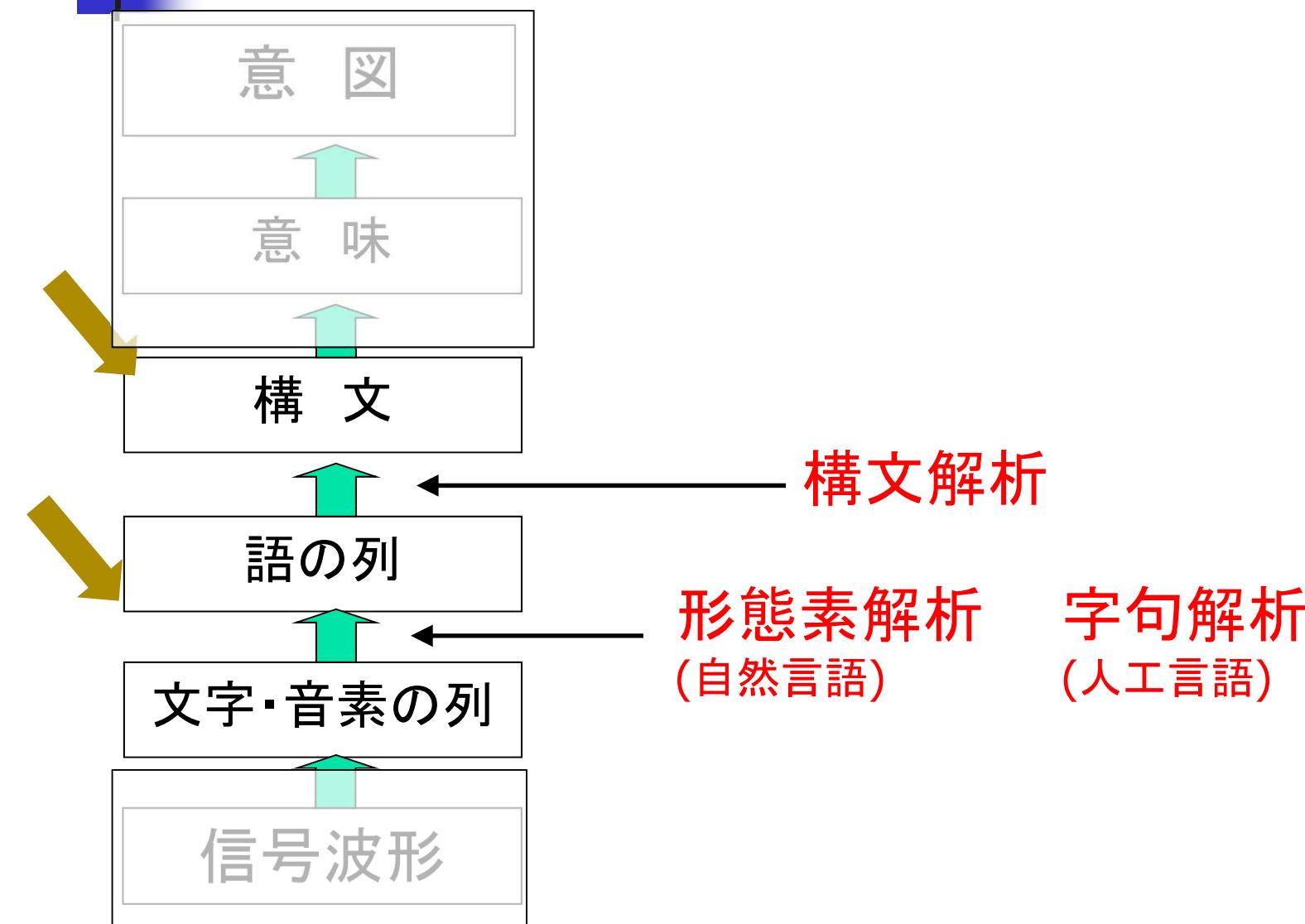
計算機科学と“言語”(1)

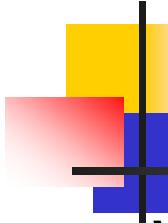
- 自然言語: 日本語, 英語, ドイツ語, ...
 - 人間が人間に意図を伝える
 - 文構造と意味は人間の知的活動中に**成立**
文法は後から構成している
 - 国際補助語 (エスペラント...)
- 人工言語(形式言語)
 - プログラミング言語(C, C++, Java, Scheme,...)
人間がコンピュータに指示する
 - マークアップ言語(HTML, XML,...)
コンピュータ間のデータ授受
 - 文構造と意味は設計され, **定義**されている

言語の構成要素と伝達



言語の処理の流れ





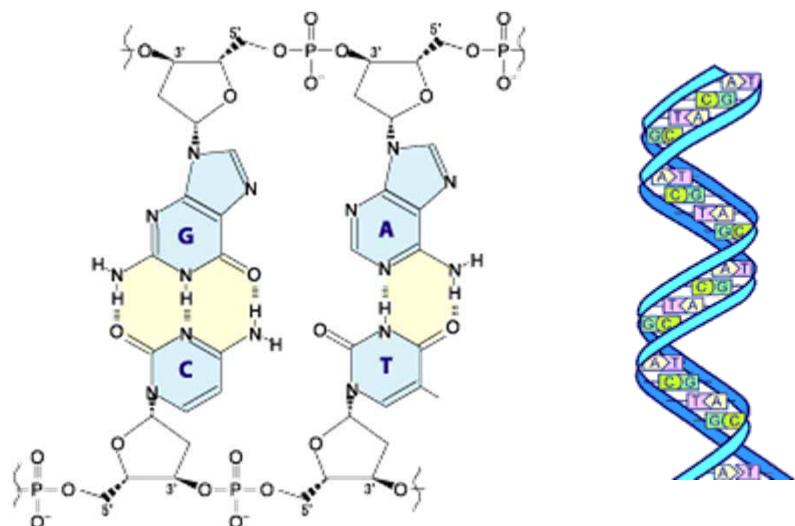
人工言語の“文”的例

```
void main()
{ init x; x = x+1; }
```

```
<TABLE><TBODY>
  <TR vAlign=center align=left>
    <TD colSpan=2>[
      <A href="http://www.i.kyoto-u.ac.jp/~akihiro/index-e.html" >English</A> |
      Japanese ]
    </TD></TR>
</TBODY></TABLE>
```

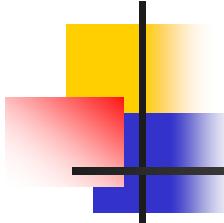
DNA

- DNA, RNA配列の類似性



Wikipediaより

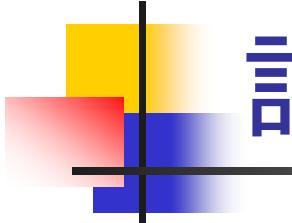
...ATTTCAC TTGGGTTAAAATG
CTGTGACCTTGAGTAAGTTGCC
GTCTCTGAATCTGATCCTTCG
ATTTC CCCATTCTCCAAACTGAG
AACTAGCACTGCTGAGACGTGG
TTATTCCAATAATAATTGTA
TATTTACATAACGCACCACAC
AACATCTCACCCAGTTGGAG
CCTACTCCTTGCTCCCGCTG...



形式言語理論

言語を文字列・単語列の集合と抽象化した上で

- 文を生成(構成・定義)するための理論
 - 正しい文字列・単語列を生成(定義)する
- 文を受理するための理論
 - 送られてきた文字列・単語列が正しいかどうかを判定する
- 句構造文法と受理機械

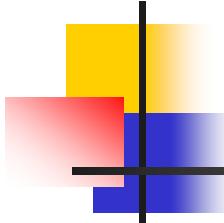


計算機科学と“言語”(1)

- ある規則性(文法)にしたがって並べられた文字または音素からなる語語(形態素)からなる文の集合
- 意味と語や文を切り離す.
 - 文法の機能に注目

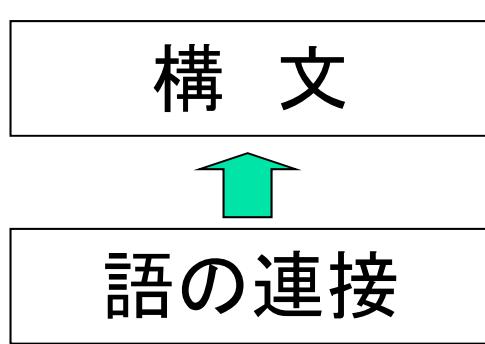


字句解析



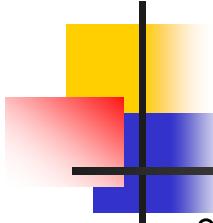
英語で記述された文の場合

- 英語で文を記述する際には、単語間に空白を入れる。(分かち書き)
→ 単語を基本単位として構文を解析すればよい



(文(名詞句(代名詞 I))(動詞句(他動詞 like)
(名詞句(固有名詞 Kyoto))))

I like Kyoto



トークン(token)

- プログラミング言語、マークアップ言語などでは、操作の最小単位を**トークン**とよぶ

例 C言語の場合

```
void main ( ) {  
    int par ;  
    par = par + 12 ;  
}
```

トークン(token)

- マークアップ言語、プログラミング言語などでは、処理の最小単位を**トークン**とよぶ

例 C言語の場合

```
void main( ) {  
    int par;  
    par = par + 12;  
}
```

トークン(token)

- プログラミング言語、マークアップ言語などでは、操作の最小単位を**トークン**とよぶ

例 HTMLの場合

- '<' と '>' で挟まれた文字列
- '</' と '>' で挟まれた文字列
(タグとよぶ)
- '&' から始まり ';' や改行で
終わる文字列
- それ以外の部分に現れる
一つずつの文字

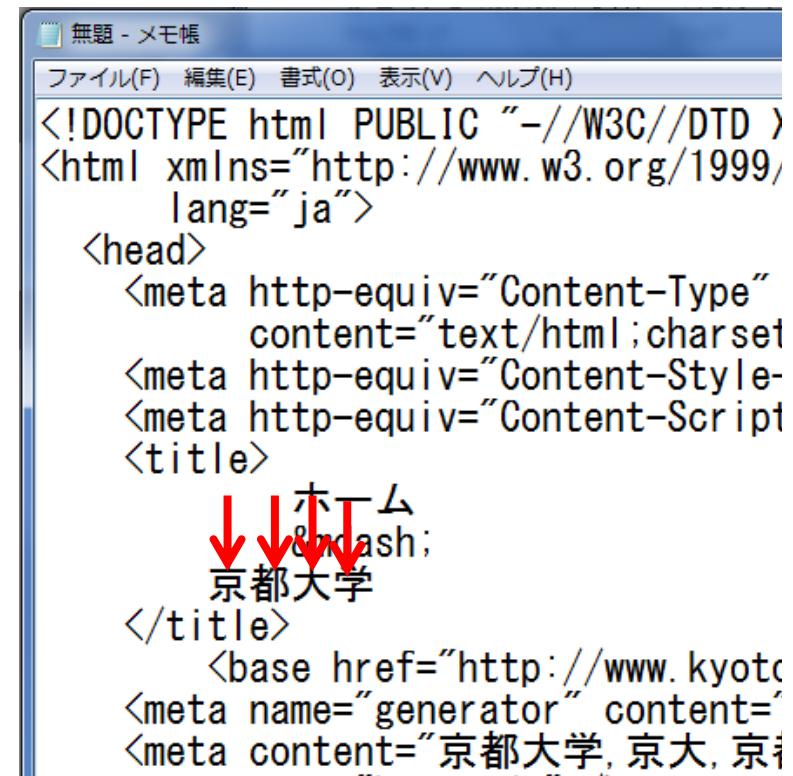
```
無題 - メモ帳
ファイル(F) 編集(E) 書式(O) 表示(V) ヘルプ(H)
<!DOCTYPE html PUBLIC "-//W3C//DTD //
<html xmlns="http://www.w3.org/1999/
    lang="ja">
<head>
<meta http-equiv="Content-Type"
      content="text/html; charset=
<meta http-equiv="Content-Style-
<meta http-equiv="Content-Script
<title>
    ホーム
    &mdash;
    京都大学
</title>
<base href="http://www.kyoto
<meta name="generator" content=
<meta content="京都大学, 京大, 京
```

トークン(token)

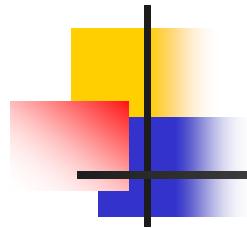
- プログラミング言語、マークアップ言語などでは、操作の最小単位を**トークン**とよぶ

例 HTMLの場合

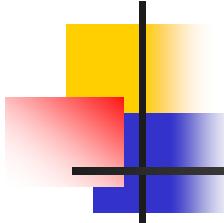
- '<' と '>' で挟まれた文字列
'</' と '>' で挟まれた文字列
(タグとよぶ)
- '&' から始まり ';' や改行で
終わる文字列
- それ以外の部分に現れる
一つずつの文字



```
無題 - メモ帳
ファイル(F) 編集(E) 書式(O) 表示(V) ヘルプ(H)
<!DOCTYPE html PUBLIC "-//W3C//DTD //
<html xmlns="http://www.w3.org/1999/
    lang="ja">
<head>
    <meta http-equiv="Content-Type"
        content="text/html; charset=
    <meta http-equiv="Content-Style-
    <meta http-equiv="Content-Script
    <title>
        ↓ ホーム
        ↓ 京都大学
    </title>
        <base href="http://www.kyoto
<meta name="generator" content=
<meta content="京都大学, 京大, 京
```

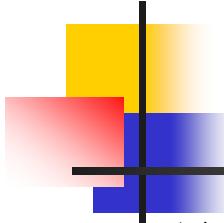


構文解析



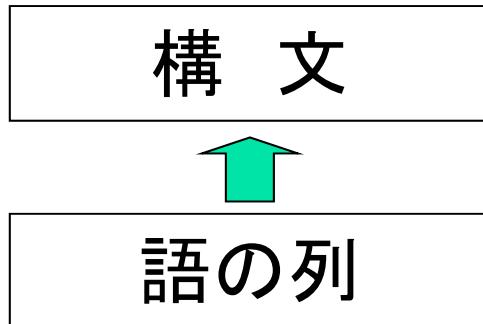
文法と構文

- 通常、ある言語において、文を単語や形態素、トークン(形態素)の列で構成するときには、ある規則に従っている。
- 規則を文法、文法に従った文が持つ構造を構文とよぶことにする



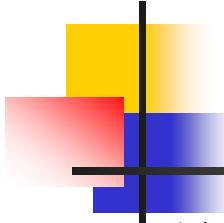
英語で記述された文の場合

- 単語の品詞と文中での位置が構文を決める。
- 構文は 単語⇒句⇒節⇒文 のように階層を成す



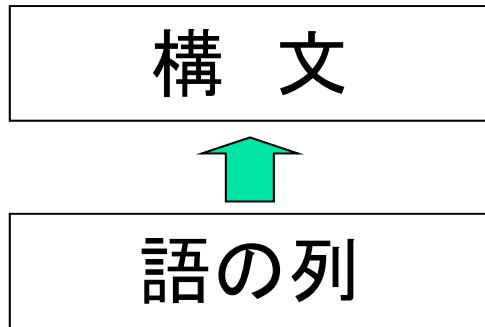
The quick brown fox jumps over the lazy dog.

冠詞 形容詞 形容詞 名詞 動詞 前置詞 冠詞 形容詞 名詞



英語で記述された文の場合

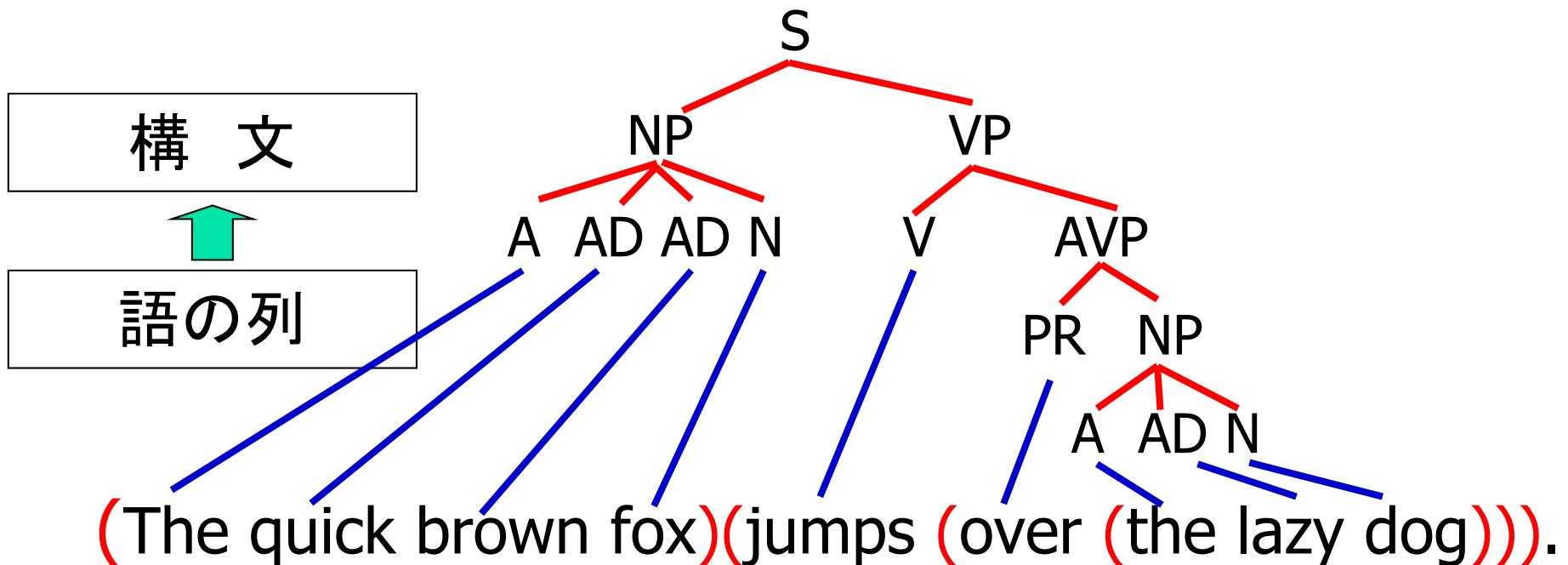
- 単語の品詞と文中での位置が構文を決める。
- 構文は 単語⇒句⇒節⇒文 のように階層を成す

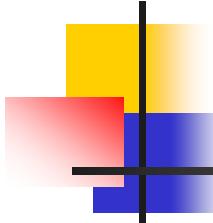


(The quick brown fox)(jumps (over (the lazy dog))).

英語で記述された文の場合

- 単語の品詞と文中での位置が構文を決める。
- 構文は 単語 ⇒ 句 ⇒ 節 ⇒ 文 のように階層を成す





句構造文法(1)

- 文法を“節や句の構造を生成するための規則”ととらえる

例 I like Kyoto.

(文(名詞句(代名詞I))(動詞句(他動詞 like)(名詞句(固有名詞 Kyoto))))



HTML(XML)風に

<文>

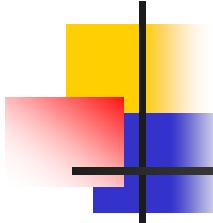
<名詞句> <代名詞> I </代名詞> </名詞句>

<動詞句> <他動詞> like </他動詞>

<名詞句> <固有名詞> Kyoto </固有名詞> </名詞句>

</動詞句>

</文>



句構造文法(2)

- 生成規則: $\alpha \rightarrow \beta$ という形の規則

例 文 → 名詞句 動詞句

名詞句 → 代名詞 動詞句 → 動詞

名詞句 → 固有名詞 動詞句 → 動詞 名詞句

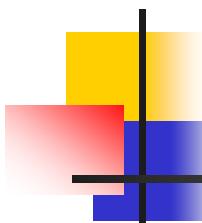
名詞句 → 冠詞 名詞 ...

...

代名詞 → I 代名詞 → you 代名詞 → he ...

固有名詞 → Kyoto 固有名詞 → Tom ...

動詞 → like 動詞 → have ...

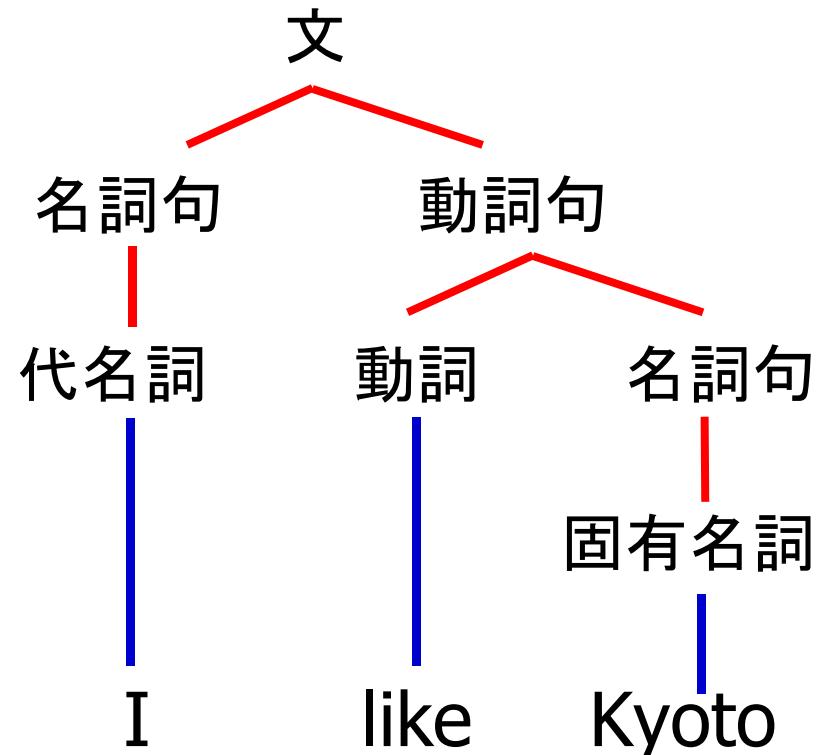


句構造文法(3)

- 生成規則 $\alpha \rightarrow \beta$ の α, β は
 - 非終端記号(構文要素)
 - 文, 名詞句, 動詞句, 代名詞, 動詞, ...
 - 終端記号(単語・文字)
- の有限列
- 文法 G : 生成規則の集合
 - と非終端記号, 終端記号, 文全体(開始)を表す非終端記号
- G によって生成される文: “文” に生成規則を有限回適用して得られる終端記号列

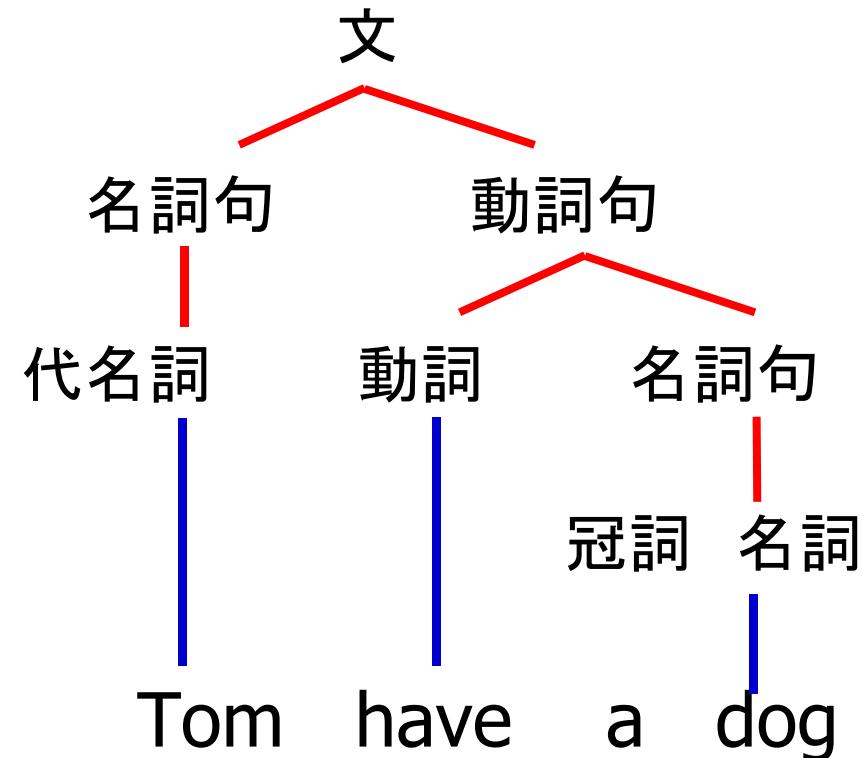
文法に従った導出の例(1)

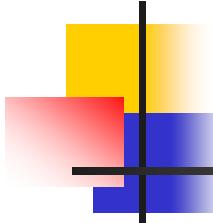
文 |— 名詞句 動詞句
|— 代名詞 動詞句
|— 固有名詞 動詞句
|— I 動詞句
|— I 動詞 名詞句
|— I like 名詞句
|— I like 固有名詞
|— I like Kyoto



文法に従った導出の例(2)

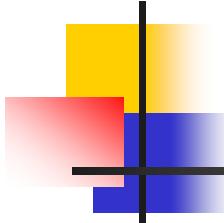
文 |— 名詞句 動詞句
 |— 代名詞 動詞句
 |— 固有名詞 動詞句
 |— Tom 動詞句
 |— Tom 動詞 名詞句
 |— Tom have 名詞句
 |— Tom have 冠詞 名詞
 |— Tom have a 名詞
 |— Tom have a bag





文法の型

- 文脈自由文法(context free): 文法中の全ての生成規則 $\alpha \rightarrow \beta$ について、 α が非終端記号一つであるもの
 - 三人称や複数の扱いを表現する際には、複雑な生成規則を用いなければならない。
 - 生成規則に条件を変えることで、様々な定義能力を持つ文法の型が導入できる。



構文解析(Parsing)

- 単語(文字)の列 σ が与えられたとき、その構文を決定すること
- 
- 単語(文字)の列 σ が与えられたとき、“文”から始まり σ で終わる導出を構成する。
 - 構文解析の結果得られる構文を表す木を**構文木**(parse tree)とよぶ

数式と文法(1)

非終端記号: 式

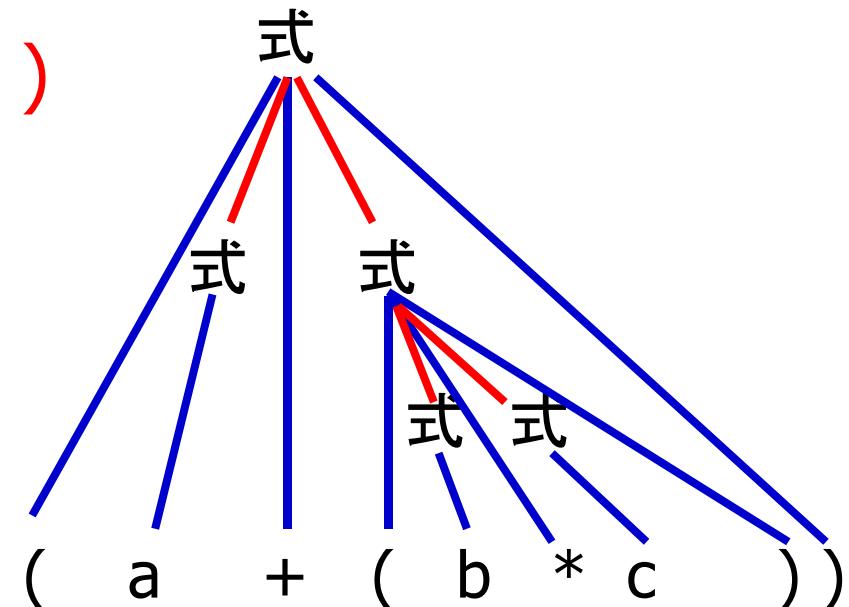
終端記号 : a, b, c, +, *, (,)

開始記号 : 式

式 \rightarrow (式 + 式)

式 \rightarrow (式 * 式)

式 \rightarrow a, b, c



式 \vdash (式 + 式) \vdash (a + 式) \vdash (a + (式 * 式))
 \vdash (a + (b * 式)) \vdash (a + (b * c))

数式と文法(2)

非終端記号：式

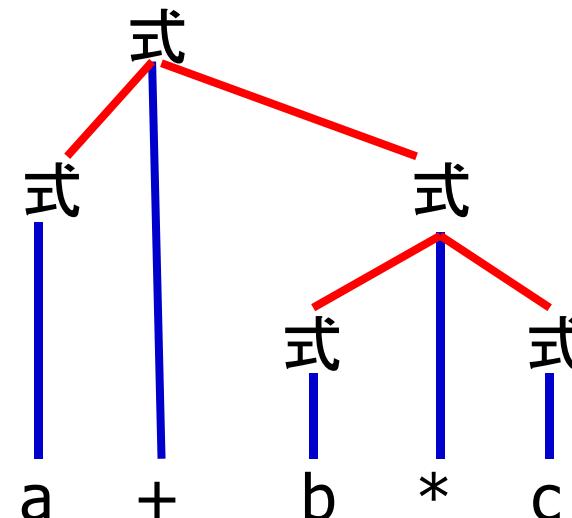
終端記号：a, b, c, +, *

開始記号：式

式 → 式 + 式

式 → 式 * 式

式 → a, b, c

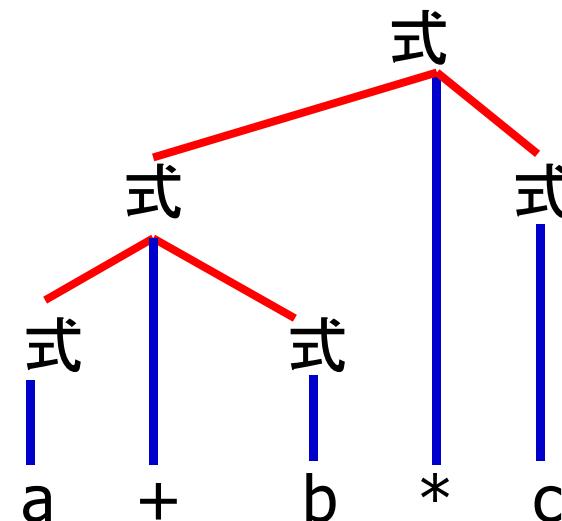
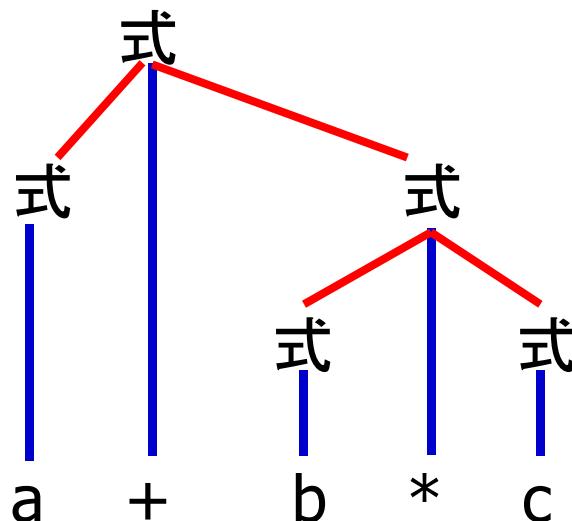


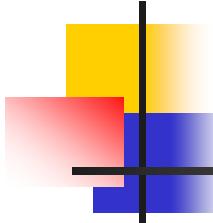
式 |- 式 + 式 |- a + 式 |- a + 式 * 式

|- a + b * 式 |- a + b * c

曖昧な文法

- ある文法を用いると、一つの単語(文字)の列 σ に対して異なる2種類以上の導出が構成されることがあるとき、その文法は**曖昧**である。





数式と文法(3)

式 → 和

和 → 和 + 和

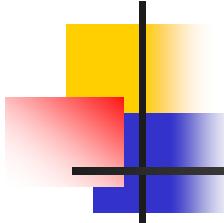
和 → 積

積 → 積 * 積

積 → 変数

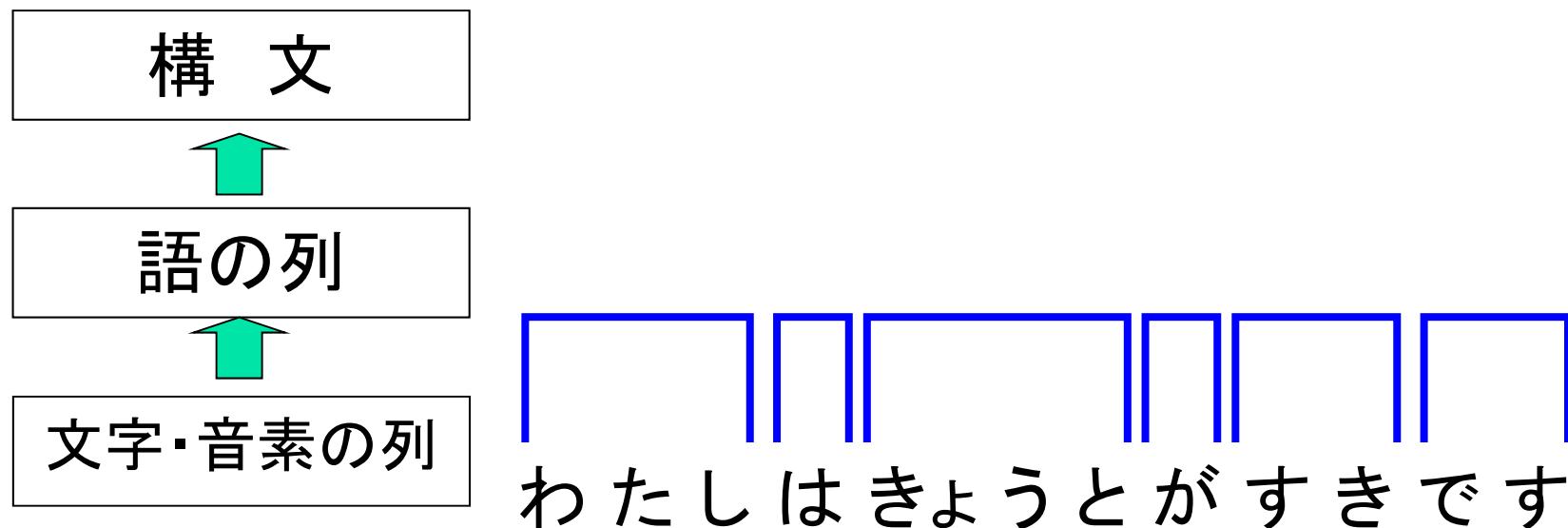
変数 → a, b, c

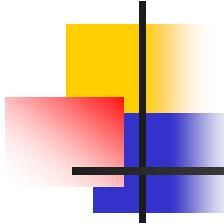
式 |
- 和 |
- 和 + 和 |
- 積 + 和 |
- 変数 + 和 |
- a + 和
- a + 積 |
- a + 積 * 積 |
- a + 変数 * 積
- a + b * 積 |
- a + b * 変数 |
- a + b * c



日本語で記述された文の場合

- 日本語で文を記述する際には、単語間に空白を入れない。
→ 単語を**推定**してから構文を解析しなければならない。





形態素(morpheme)

- 文字列(音素列)において意味を持つ最小単位
 - それ以上分解すると意味をなさなくなる

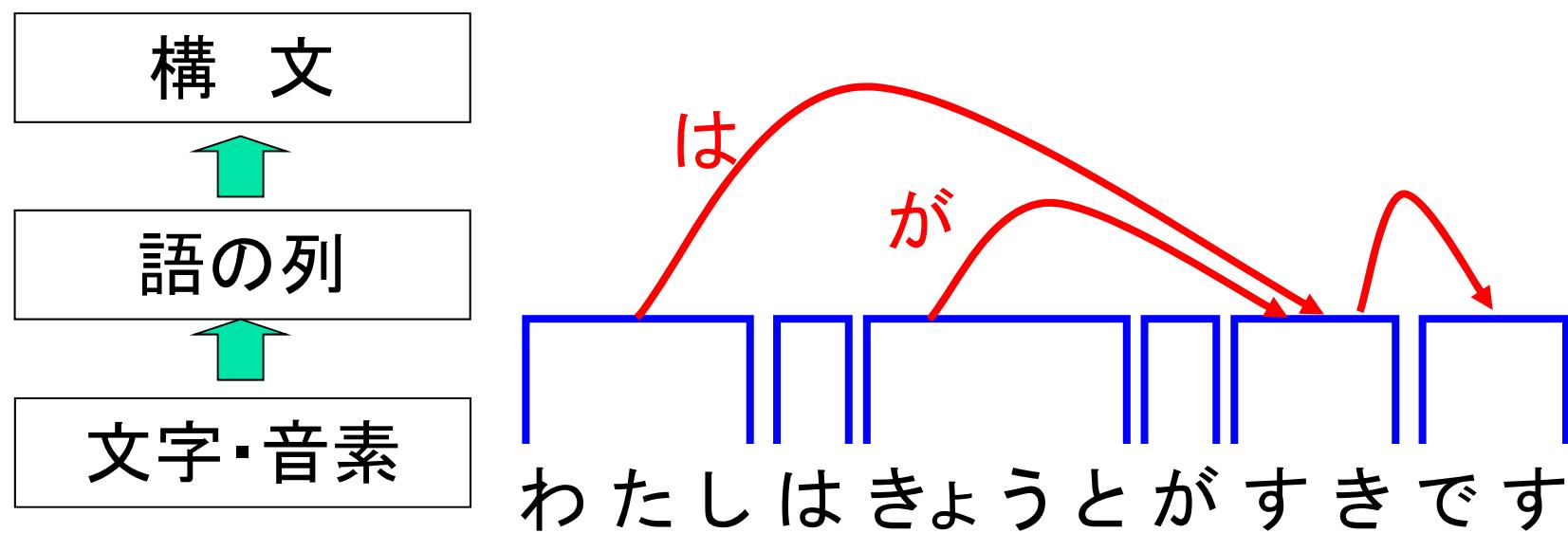
例 わたしはきょうとにりょこうする
すももももももものうち

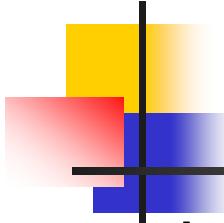
（各言葉を構成する形態素が青い線で囲まれている）

- 自然言語については、一つの言語に対して、複数の文法が考案されている。
→ “単語”の定義は文法に依存する
- 英語でも、接頭辞(un-, non-など)や接尾辞(-tion, -ible)などは一つの形態素として扱う

日本語で記述された文の場合

- 単語や句の修飾・被修飾関係が構文を決めている





プログラミング言語の場合

- 数式と同様の構文規則を用いる
- トークン自体にも構文規則がある

例 正整数の10進表現 → 0以外の数字 数字列

数字列 → 数字 数字列

数字 → 0

数字 → 0以外の数字

0以外の数字 → 0

0以外の数字 → 1

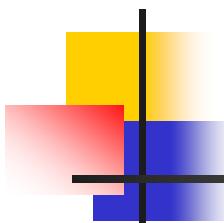
0以外の数字 → 2

0以外の数字 → 3

0以外の数字 → 4

...

0以外の数字 → 9



マークアップの場合(1)

DTD宣言→`<!DOCTYPE DTD属性列>`

文書開始タグ→`<html 文書属性列>`

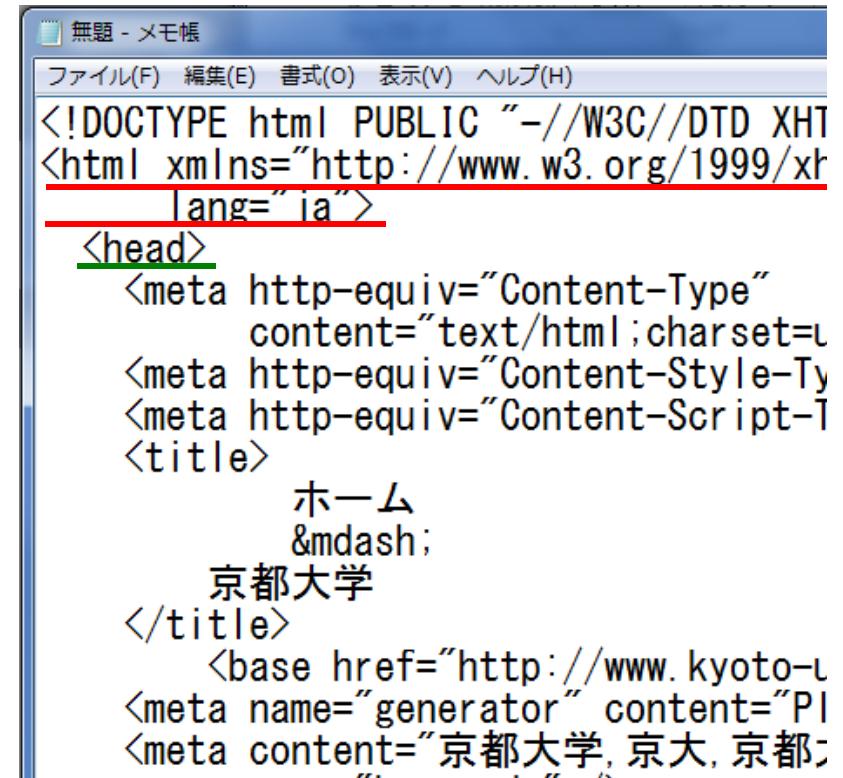
文書終了タグ→`</html>`

頭部開始タグ→`<head>`

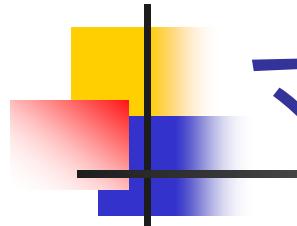
頭部終了タグ→`</head>`

本体開始タグ→`<body>`

本体終了タグ→`</body>`



```
無題 - ノート帳
ファイル(F) 編集(E) 書式(O) 表示(V) ヘルプ(H)
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
<html xmlns="http://www.w3.org/1999/xhtml"
      lang="ja">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
  <meta http-equiv="Content-Style-Type" content="text/css"/>
  <meta http-equiv="Content-Script-Type" content="text/javascript"/>
  <title>
    ホーム
    &mdash;
    京都大学
  </title>
  <base href="http://www.kyoto-u.ac.jp/"/>
  <meta name="generator" content="HTML Tidy 4.5.2, see <a href="http://tidy.sourceforge.net/">tidy.sourceforge.net</a>">
  <meta content="京都大学, 京大, 京都" name="description"/>
  <meta content="京都大学の公式ウェブサイト" name="keywords"/>
```



マークアップの場合(2)

文書 → DTD宣言 文書開始タグ 頭部 本体 文書終了タグ

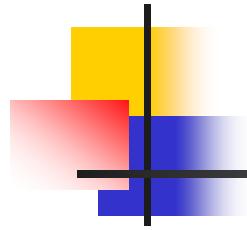
頭部 → 頭部開始タグ タグ列 タイトル部 タグ列

頭部終了タグ

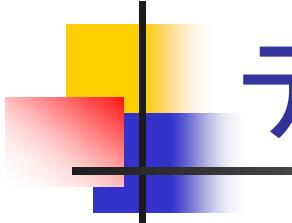
タイトル部→タイトル開始タグ 文字列 タイトル終了タグ

本体 → 本体開始タグ タグと文字列 頭部終了本体

- 直観的には、数式の括弧‘(’, ‘)’がそれぞれ開始タグ、終了タグに対応し、定数(変数)が文字列に対応する

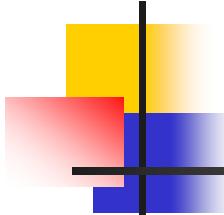


機械学習と形式言語理論



テキストマイニング

- 人工知能(機械学習)データマイニング技術、統計学を文書に対して適用することにより分析を行う手法を“テキストマイニング”という。
 - 機械学習:元来は、人工知能に持たせる“学習する機能”を意味していたが、現在では統計的データ解析アルゴリズム全般を指している。
 - データマイニング:大量のデータに潜む隠れた規則性を発見すること



単語の統計を利用する

- 最近のニュース記事を集めて、ニュースのジャンル別に出現する単語の頻度を計測する

例 1: 首相 が 党首 討論: 政治

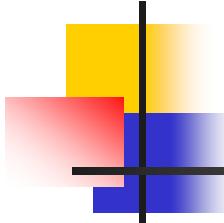
2: 安倍 首相 が 会談: 政治

3: 阪神 が 連勝: スポーツ

4: 日経 平均 が 反落: 経済

5: 首相 が 日銀 総裁 と 会談: 経済

記事番号	首相	が	党首	...	連勝	...	日経	...	クラス
1	1	1	1		0		0		政治
2	1	1	0		0		0		政治
3	0	1	0		1		0		スポーツ
4	0	1	0		0		1		経済
5	1	1	0		0		0		経済



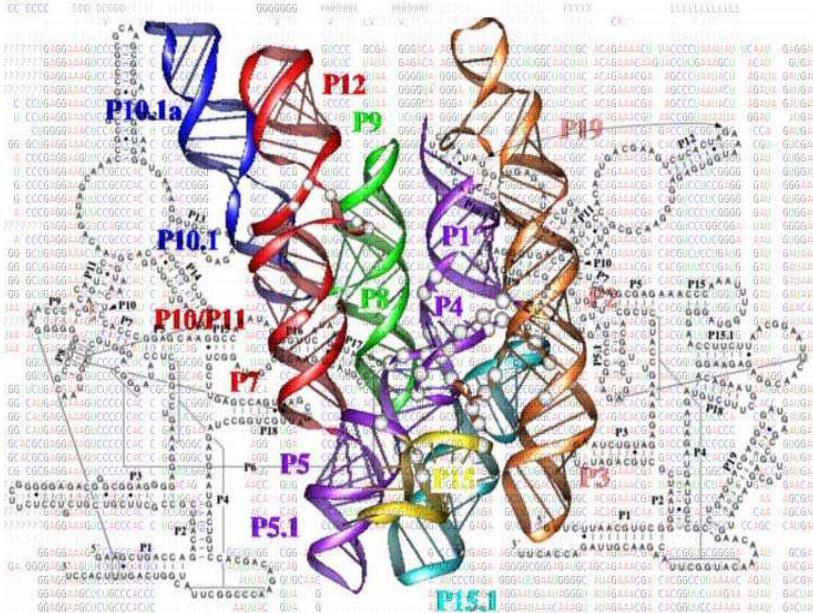
文書の自動分類

- 新しいニュース記事のジャンルを自動で分類
 - “迷惑メールのフィルタリング”で有名になった

例 もし“首相”という単語を含む記事が新たに来たときに、そのジャンルは何か？

記事番号	首相	が	党首	...	連勝	...	日経	...	クラス
1	1	1	1		0		0		政治
2	1	1	0		0		0		政治
3	0	1	0		1		0		スポーツ
4	0	1	0		0		1		経済
5	1	1	0		0		0		経済
...									

RNA Classification



Non-coding RNA (ncRNA) is an important molecule in bioinformatics, and is considered to be a factor of the difference between higher organism and others

RNA sequences are accumulated in **RNA families**, and the members in each family have similar structures and functions. ([Rfam database](#))

... - Microsoft Internet Explorer

お問い合わせ サポート ヘルプ

sanger.ac.uk/Software/genomes/

検索 ブックマーク フォルダ 21 ナビゲーション RNA family

RNA families database of alignments and CMs

Wellcome Trust Sanger Institute

Search Sequence Search Browse Align Generators np Help miRNA

involve researchers based at the [Wellcome Trust Sanger Institute](#), Cambridge, UK and Janelle Farm. It is a large collection of multiple sequence alignments and covariance models covering many common for each family in Rfam.

add multiple sequence alignments
notation
distribution of family members
hendatabasel

[INFERNAL](#) software suite. Rfam can be used to annotate sequences (including complete genomes) in non-coding RNAs. Please read important information about [using Rfam for genome annotation](#). We list putative RNAs in over 200 [complete genomes](#), and a [web search facility](#) for short sequences.

ge amount of available data, especially published multiple sequence alignments, and repackages these and suitable resources. We have made many efforts to include individual sources on family pages, of links to these sources [here](#). If you find any of the data presented here useful, please also be sure to cite it.

Rfam, and using this site, click [here](#).

Worldwide [FTP access to Rfam](#)

You can also download the Rfam database and for instance search in locally using the [INFERNAL](#) covariance model software. [Hotline](#) directly to the Rfam site or [View Rfam site files](#)

Enter your keyword(s) here

Version 8.0

February 2007, 574 families

Enter an EMBL name or accession number

Enter your keyword(s) here

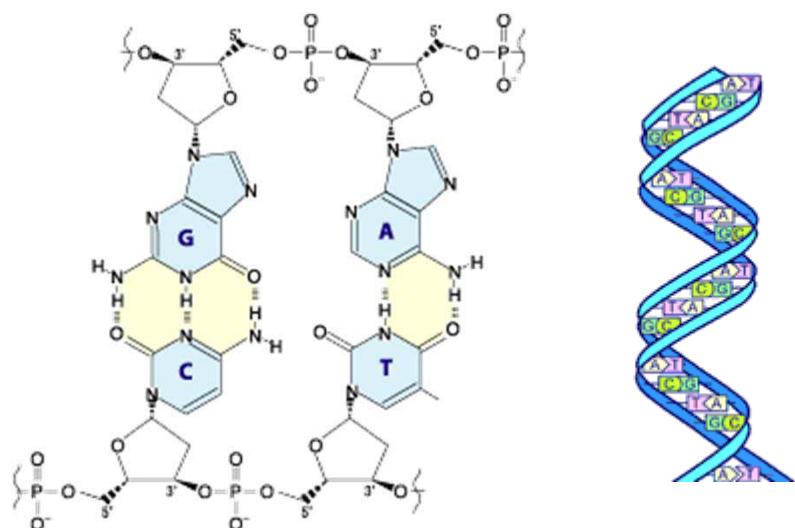
Go Example

Enter an EMBL name or accession number

Go Example

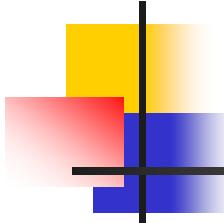
DNA

- DNA, RNA配列の類似性



Wikipediaより

...ATTTCAC TTGGGTTAAAATG
CTGTGACCTT GAGTAAGTTGCC
GTCTCTGAATCTGATCCTTTCG
ATTTC CCCATTCTCCAAACTGAG
AACTAGCACTGCTGAGACGTGG
TTATTCCAATAATAATTGTA
TATTTACATAACGCACCACAC
AACATCTCACCCAGTTGGAG
CCTACTCCTTGCTCCCGCTG...



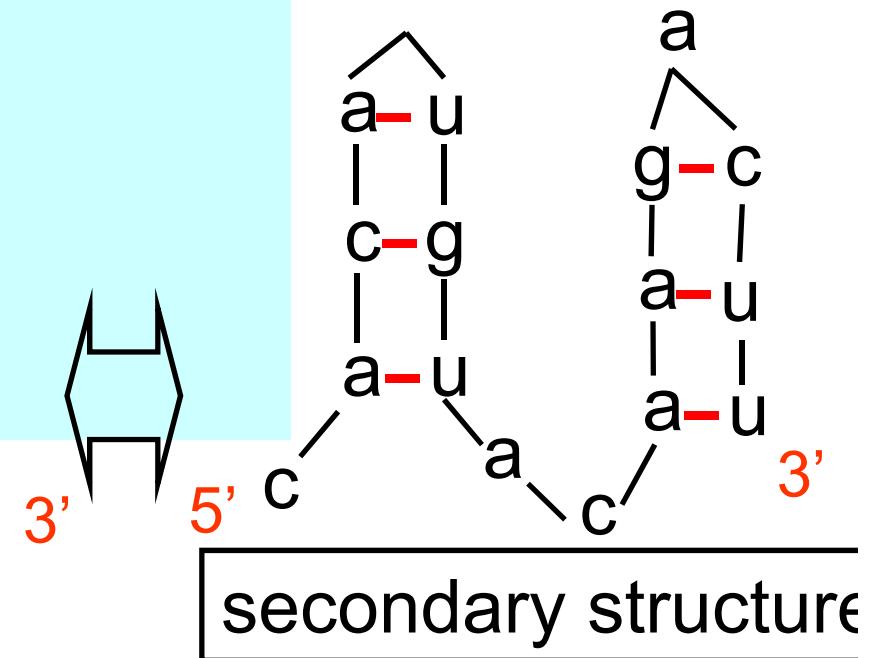
データモデリングの例: 蛋白質

- 一次構造: アミノ酸の配列
- 二次構造: α -ヘリックス構造、 β シート構造
- 三次構造: 立体構造
- 四次構造: タンパク質が数個集まってできる構造

The **secondary structures** detected by base pairs (**a-u, c-g**) are important in RNA classification.

5' c a c a u g u a c a a g a c u u

RNA sequence



Our purpose

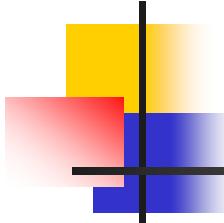
To distinguish between the member sequences in a given **RNA family** and non-member sequences.

State of art in this field

SVM combined with kernel functions for structured data.
e.g. string kernel which

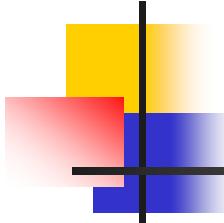


記号列の数学



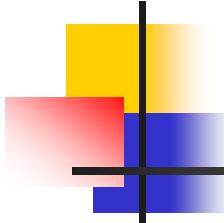
記号列の数学への誘い

- 列は計算機科学の基本をなしている
 - 1と0の列
 - ASCII文字の列
 - ひらがな,漢字の列
 - 単語の列
 - A, T, C, Gの列
- 四則演算のような強力な演算がない
 - 連接: $s \cdot t$ 記号列 s の後に t をつなげる
結合律: $s \cdot (t \cdot u) = (s \cdot t) \cdot u$
単位元: 空の記号列 $\varepsilon \cdot s = s \cdot \varepsilon = s$
 - 演算をアルゴリズムで定義する



形式言語理論では

- 操作の最小単位を記号または文字と呼ぶ
 - 英単語から構成される文を議論するときには、各英単語を記号もしくは文字として扱う
 - トークンからなる式を議論するときには、トークンを記号もしくは文字として扱う
 - トークンの構成を議論するときには、数字や英文字を各英単語を記号もしくは文字として扱う
- 対象とする記号または文字の有限集合をアルファベット(alphabet)とよぶ
- 有限長の記号からなる列を記号列または語(word)とよぶ



記号列データ

- Σ : アルファベット(alphabet)
- Σ^* : Σ 中の記号からなる有限列全体の集合
 - 空列を ε で表す
 - $\Sigma^+ = \Sigma^* - \{\varepsilon\}$

Example 1

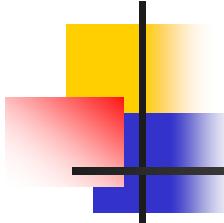
$$\Sigma = \{a, b\}$$

$$\Sigma^* = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, \dots, abaabab, \dots\}$$

Example 2

$$\Sigma = \{A, T, C, G\}$$

$$\Sigma^* = \{\varepsilon, A, T, C, G, AA, AT, AC, AG, \dots, AAA, AAT, \dots\}$$



列と系列

- 列 : string 系列 : sequence
- 部分列(部分語) : substring (subword)
部分系列 : subsequence

例 $w = abaab$

ba, baa などは w の部分列(部分語) であり,

部分系列でもある

aaa や bbは w の部分列(部分語) ではないが,

部分系列である

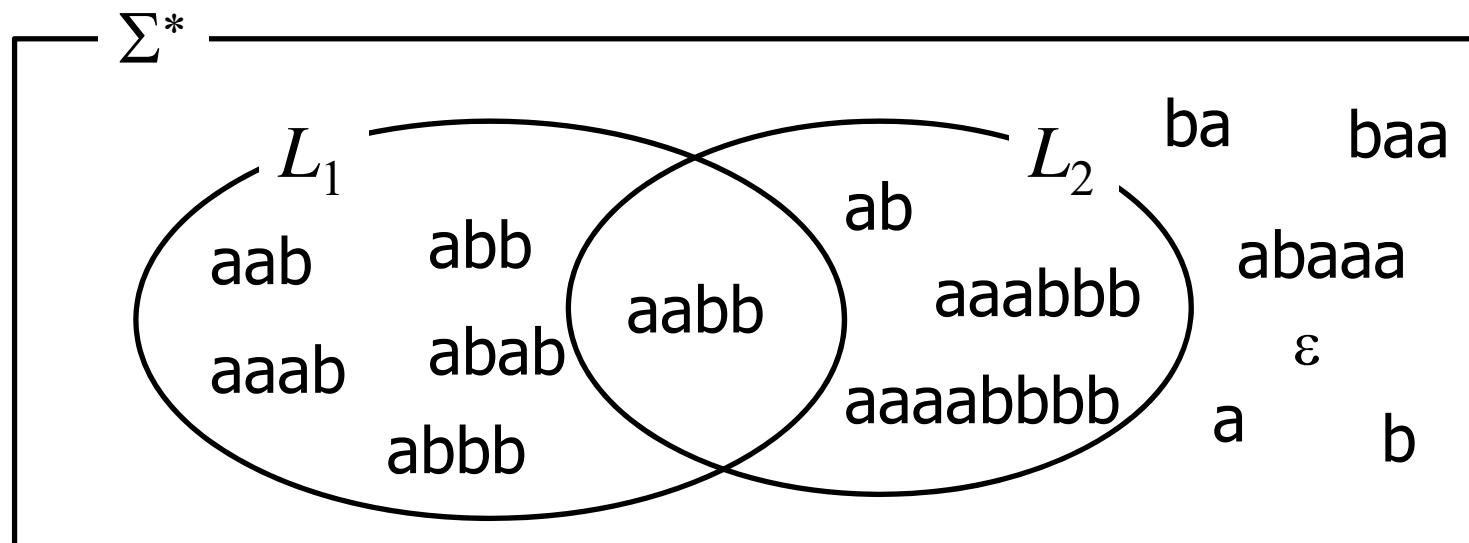
形式言語

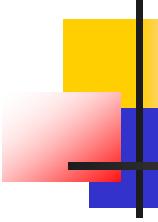
- 全体集合を Σ^* とするとき、 Σ^* の部分集合を**形式言語**(あるいは**単に言語**)とよぶ。
 - 以後、アルファベット Σ が文脈から明らかなときは、単に**記号列の集合**とよぶこともある。

例 $\Sigma = \{a, b\}$, $\Sigma^* = \{\epsilon, a, b, aa, ab, ba, bb, aaa, aab, \dots\}$

$L_1 = \{aab, abb, aaab, aabb, abab, abbb, \dots\}$

$L_2 = \{ab, aabb, aaabbb, aaaabbbb, \dots\}$





集合の内包的表現と規則性

集合を表す2種類の方法

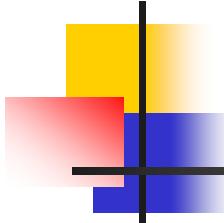
- 要素を並べて表す(外延的)

$$\Sigma = \{a, b\}$$

$$L = \{ab, aab, abab, aaab, abaab, aaaabb, \dots\}$$

- 全体集合の要素がその集合に属する**条件**を用いて表す(内包的)

$$L = \{w \in \Sigma^* \mid w \text{ starts with } a \text{ and ends with } b\}$$



用語と記号(1)

アルファベット Σ を一つ固定する.

- 記号列 $w = c_1c_2\dots c_n$ の長さを $|w| = n$ と定義する.

空列 ε に対しては $|\varepsilon| = 0$ と定義する.

例 $|abb| = 3, |abaabab| = 7$

- 形式言語 L と自然数 n に対して形式言語 $L|_n$ を

$L|_n = \{w \in \Sigma^* \mid w \in L \text{かつ } |w| = n\}$ と定義する.

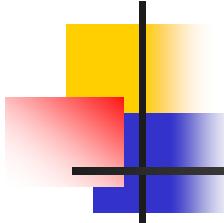
- 形式言語 L, M に対して形式言語 $L \cdot M$ を

$L \cdot M = \{uv \in \Sigma^* \mid u \in L \text{かつ } v \in M\}$ と定義する.

■ 演算子 \cdot はしばしば省略することがある.

例 $L = \{ab, abb, ba\}, M = \{\varepsilon, bb, bab\}$ とするとき

$L \cdot M = \{ab, abb, ba, abbb, abbbb, babb, ba, babb, babab\}.$



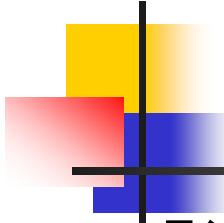
命題と注意

命題 アルファベット Σ を一つ固定する.

形式言語 L, M, N に対して $(L \cdot M) \cdot N = L \cdot (M \cdot N)$.

したがって、 $(L \cdot M) \cdot N$ を $L \cdot M \cdot N$ と表記する.

注意 一般には $L \cdot M \neq M \cdot N$.



用語と記号(2)

- 形式言語 L と非負整数 n に対して形式言語 L^n を以下のように帰納的に(再帰的に)定義する。
 - $L^0 = \{\varepsilon\}$. ($L^0 \neq \emptyset$ に注意)
 - $L^1 = L$.
 - $L^{n+1} = L \cdot L^n$.

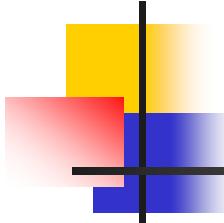
注意1. 形式言語理論では L^n は L の n 回の直積 $L \times L \times \cdots \times L$ ではない。

注意2. $L \cdot L^n = L^n \cdot L$ が成り立つ。

例 $L = \{ab, abb, ba\}$, $M = \{\varepsilon, bb, bab\}$ とするとき

$$L^2 = \{abab, ababb, abba, abbab, abbabb, abbba, \\ baab, baabb, baba\},$$

$$M^3 = \{\varepsilon, bb, bab, bbbb, bbbab, babbab, babb, babbab, \\ bbbbbbab, bbbabbab, bbbabbb, bbbabbab \\ babbabbab, babbabbab, babbabbb, babbabbab\}$$



用語と記号(3)

- 形式言語 L に対して L^* を

$$L^* = L^0 \cup L^1 \cup \dots \cup L^n \cup \dots$$

$$= \{w \in \Sigma^* / \text{ある非負整数 } n \text{ が存在して } w \in L^n\}$$

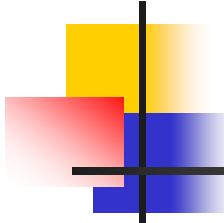
と定義する。

例 $L = \{ab, abb\}$ とするとき

$$\begin{aligned} L^* = & \{\varepsilon, ab, abb, abab, ababb, abbab, abbabb, \\ & ababab, abababb, ababbab, ababbabb, \\ & abbabab, abbababb, abbabbab, abbabbabb, \\ & abababab, ababababb, abababbab, abababbabb, \\ & ababbabab, ababbababb, abbababbab, abbababbabb, \\ & abbabbabab, abbabbababb, abbabbabbab, abbabbabbabb, \dots\} \end{aligned}$$

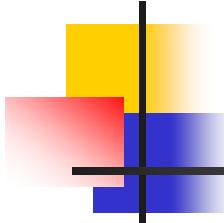


受講上の注意



計算機科学 v.s. 数学

- 理論の構成方法は数学の構成方法を規範としている
 - 概念の数学的定義, 定理, 証明, ...
- 論法は数学のそれとは似て非なるもの



論法の違いの例

定理 非負整数 m, n には最大公約数 d が存在する.

証明1 非負整数 k の約数全体の集合を $I(k)$ と表すとき, 集合 $I(m) \cap I(n)$ は m と n の公約数の集合である. この集合は自然数の集合だから, 必ず最小値 d が存在する.

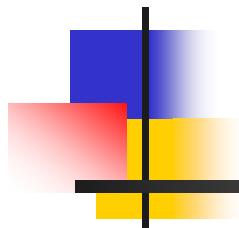
証明2 非負整数 m を n で割った商を q , 余りを r とすると,

$$m = nq + r_0 \quad (0 \leq r < n)$$

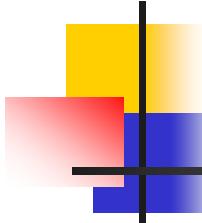
だから, m と n の公約数は r の約数でなければならぬ.

また n と r の公約数は m の約数でなければならないから, m と n の最大公約数は n と r の最大公約数に一致する. 次に n を q で割った余り r_1 に対しても同様のことが成立する. こえを繰り返して r_0, r_1, r_2, \dots を次々求めれば, この数列は単調減少列だから必ず有限列になり, その最後の項が m と n の最大公約数である.

言語・オートマトン 有限状態オートマトン



山本章博
情報学研究科 知能情報学専攻
(工学部 情報学科)
2017年版

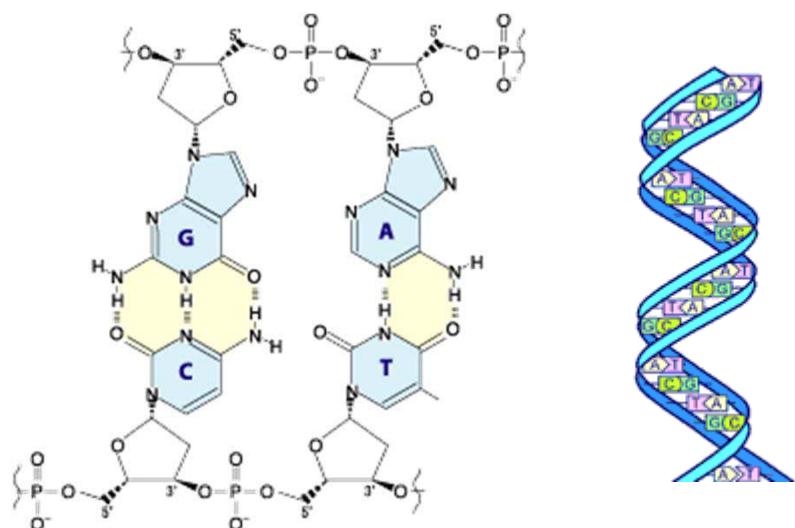


自然言語の“文”的例

- 私は京都が好きです.
 - 京都は私のお気に入りです.
 - 京都って、いいよね.
-
- I like Kyoto.
 - Kyoto is one of my favorite cities.
 - Kyoto, my favorite!

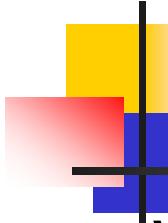
DNA

- DNA, RNA配列の類似性



Wikipediaより

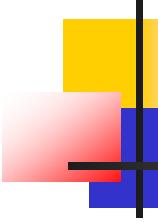
...ATTTCAC TTGGGTTAAAATG
CTGTGACCTTGAGTAAGTTGCC
GTCTCTGAATCTGATCCTTCG
ATTTC CCCATTCTCCAAACTGAG
AACTAGCACTGCTGAGACGTGG
TTATTCCAATAATAATTGTA
TATTTACATAACGCACCACAC
AACATCTCACCCAGTTGGAG
CCTACTCCTTGCTCCCGCTG...



人工言語の“文”的例

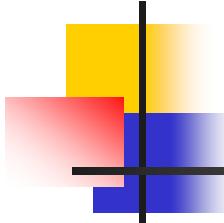
```
void main()
{ init x; x = x+1; }
```

```
<TABLE><TBODY>
  <TR vAlign=center align=left>
    <TD colSpan=2>[
      <A href="http://www.i.kyoto-u.ac.jp/~akihiro/index-e.html" >English</A> |
      Japanese ]
    </TD></TR>
</TBODY></TABLE>
```



計算機科学における“言語”

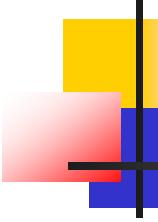
- ある規則性(文法)にしたがって並べられた文字または音素からなる語語(形態素)からなる文の集合
- 意味と語や文を切り離す.
 - 文法の機能に注目



形式言語理論

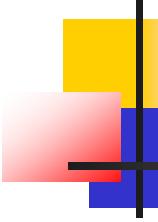
言語を記号列・単語列の集合と抽象化した上で

- 文を生成(構成・定義)するための理論
 - 正しい記号列・単語列を生成(定義)する
- 文を受理するための理論
 - 送られてきた記号列・単語列が正しいかどうかを判定する
- 句構造文法と受理機械



記号列の数学

- 列は計算機科学の基本をなしている
 - 1と0の列
 - ASCII記号の列
 - ひらがな,漢字の列
 - 単語の列
 - A, T, C, Gの列
- 四則演算のような強力な演算がない
 - 連接: $s \cdot t$ 記号列 s の後に t をつなげる
結合律: $s \cdot (t \cdot u) = (s \cdot t) \cdot u$
単位元: 空の記号列 ε (λ)
 - 演算をアルゴリズムで定義する



記号列

- Σ : 記号の有限集合, アルファベット(alphabet)とよぶ
- Σ^* : Σ 中の記号からなる有限列全体の集合
 - 空列を ε で表す
 - $\Sigma^+ = \Sigma^* - \{\varepsilon\}$

Example 1

$$\Sigma = \{a, b\}$$

$$\Sigma^* = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, \dots\}$$

Example 2

$$\Sigma = \{A, T, C, G\}$$

$$\Sigma^* = \{\varepsilon, A, T, C, G, AA, AT, AC, AG, \dots, AAA, AAT, \dots\}$$

形式言語

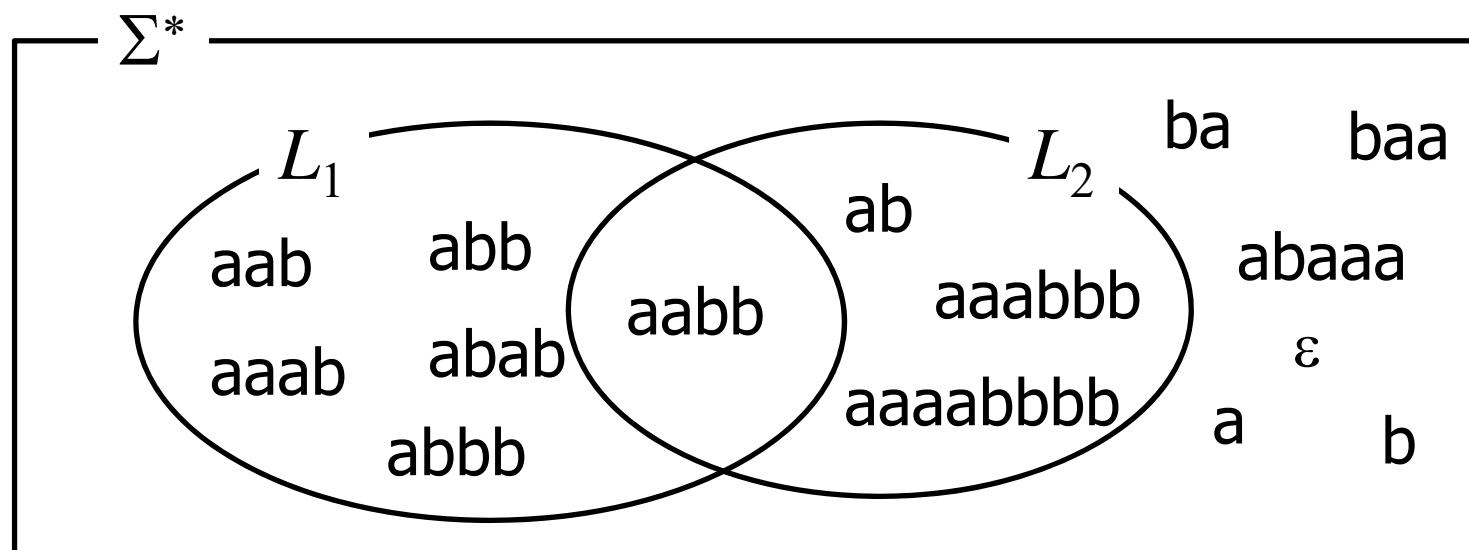
- 全体集合 Σ^* の部分集合を**形式言語**とよぶ.

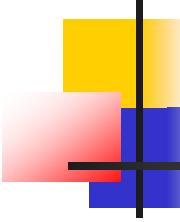
Example

$$\Sigma = \{a, b\}, \Sigma^* = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, \dots\}$$

$$L_1 = \{aab, abb, aaab, aabb, abab, abbb, \dots\}$$

$$L_2 = \{ab, aabb, aaabbb, aaaabbbb, \dots\}$$





形式言語と規則

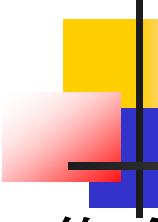
- 言語である以上、正しい“語”または“文”を定める規則がある。

例 あるプログラミング言語の変数

例 XML言語のタグ

例 日本語単語の綴りにおける“ん”的使い方

例 英単語の綴りにおけるqの使い方



集合の内包的表現と規則性

集合を表す2種類の方法

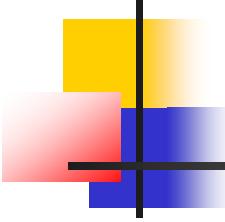
- 要素を並べて表す(外延的)

$$\Sigma = \{a, b\}$$

$$L = \{ab, aab, abab, aaab, abaab, aaaabb, \dots\}$$

- 全体集合の要素がその集合に属する**条件**を用いて表す(内包的)

$$L = \{w \in \Sigma^* \mid w \text{ starts with } a \text{ and ends with } b\}$$



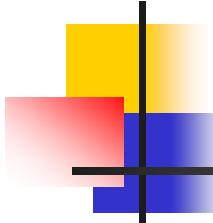
形式言語と文法

- 言語である以上、正しい“語”または“文”を定める規則がある。
- 規則の表現方法
 - 受理機械
 - 式
 - 句構造文法

0型	句構造文法	Turing Machine
1型	文脈依存文法	線形有界オートマトン
2型	文脈自由文法	非決定性プッシュダウン・オートマトン
3型	正則文法	有限状態オートマトン



有限状態機械と計算



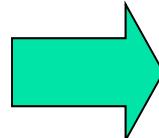
“計算”を人工的に作る

- コンピューター(computer)は、人間が行う“計算”という行為をシミュレートする研究がもとになっている。

$$(((2 \times 3) + (4 \times 6)) + (7 \times 8)) \div (2 + 4)$$



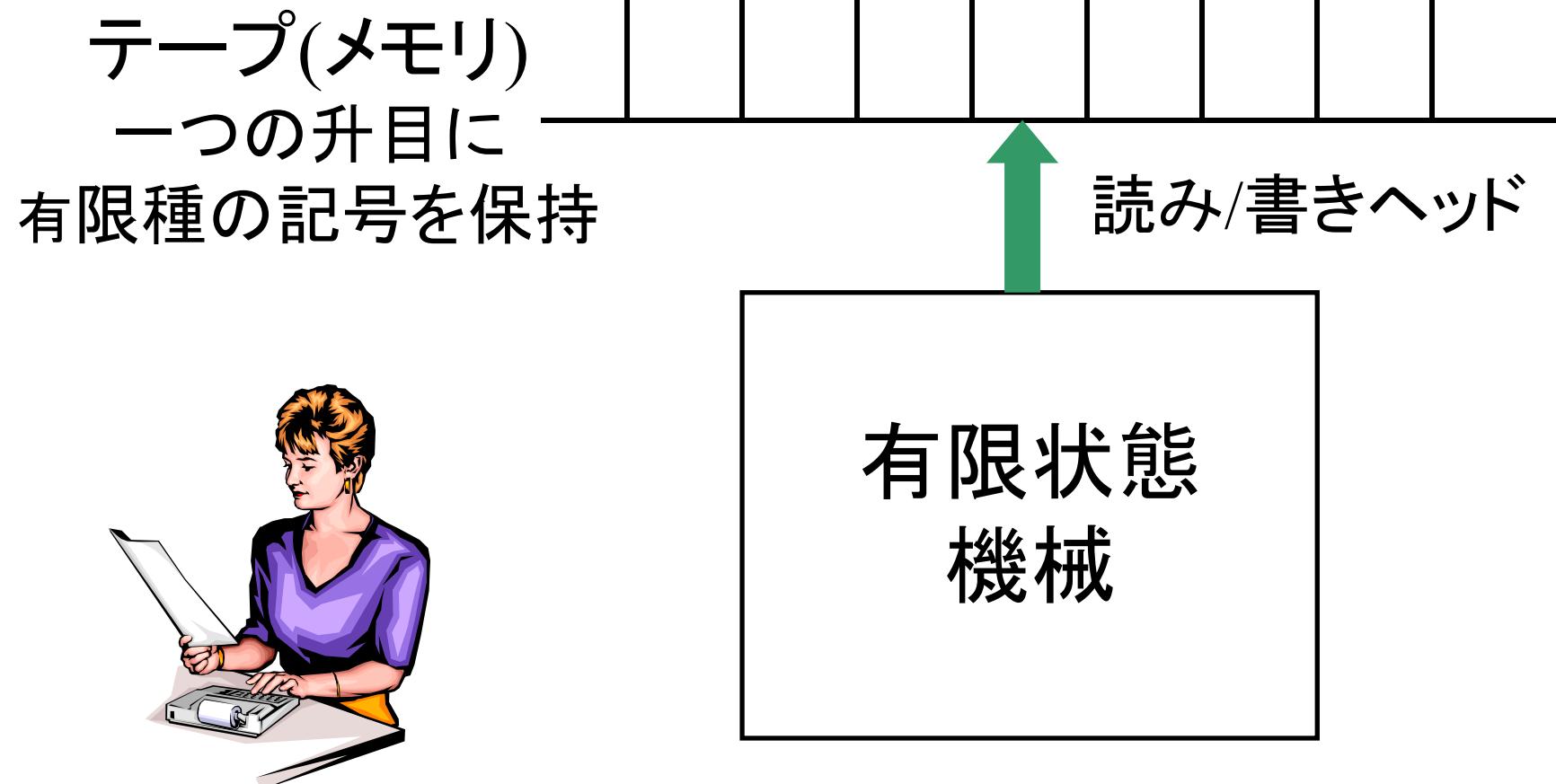
computer (計算手)



computer (計算機)

Turing Machine(1936)

Alan Turingが考案





2012 THE ALAN TURING YEAR

A Centenary Celebration of the Life and Work of Alan Turing

Centenary Events

- ATY EVENTS OVERVIEW
- ATY EVENTS CALENDAR
- ATY EVENTS A4 HANDOUT
- ATY RESOURCES
- TCAC Arts & Culture Subtee
- TCAC Media Group
- Turing Manchester 2012
- TCAC Manchester
- Alan Turing Jahr 2012
- TCAC Germany
- Alan Turing Jaar 2012
- AAAI Turing Lecture New
- ACE 2012, Cambridge
- ACM Centenary Celebration
- AI at Donetsk, Ukraine
- AI*IA Symp. Artificial Intelligence
- Alan Mathison Turing, Roma
- Alan Turing Centenary in Calgary
- ALAN TURING CONF, Manchester
- Alan Turing Days in Lausanne
- AMS Special Session, USA
- AMS-ASL Joint Math Meeting
- Animation12, Manchester

• To link to this webpage please use the url: <http://www.turingcentenary.eu/> - and add the ATY logo (suitably resized) to your webpage. See also [pdf version](#) or [monochrome version](#)

• If you wish to be included in the Turing Centenary email list, please enter your email address here and press Submit:

The Alan Turing Year on Facebook - and on Twitter

ATY Press and Media Contact - [Daniela Derbyshire](mailto:Daniela.Derbyshire@turing.co.uk) - email: turing @ live.co.uk



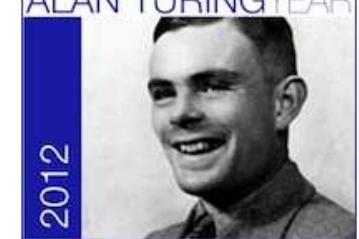
The Turing Test
a one-hour opera, sung in English
UK tour 2012 - help us make it happen!

June 23, 2012, is the Centenary of Alan Turing's birth in London. During his relatively brief life, Turing made a unique impact on the history of computing, computer science, artificial intelligence, developmental biology, and the mathematical theory of computability.

ALAN TURINGYEAR

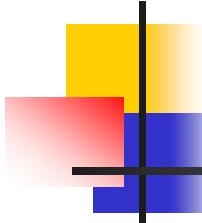
2012



News

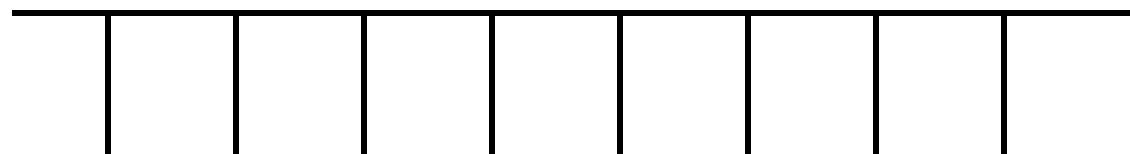
- 19.04.12**
GCHQ releases two codebreaking papers by Alan Turing
- 09.04.12**
The biography of Alan M Turing by his mother Sara appears
- 06.04.12**
Manchester Pride Festival to honour Alan Turing

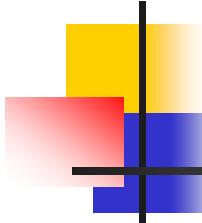
- A. Turing: On computable numbers, with an application to the Entscheidungsproblem, 1936.



Turingによる計算の機械化(1)

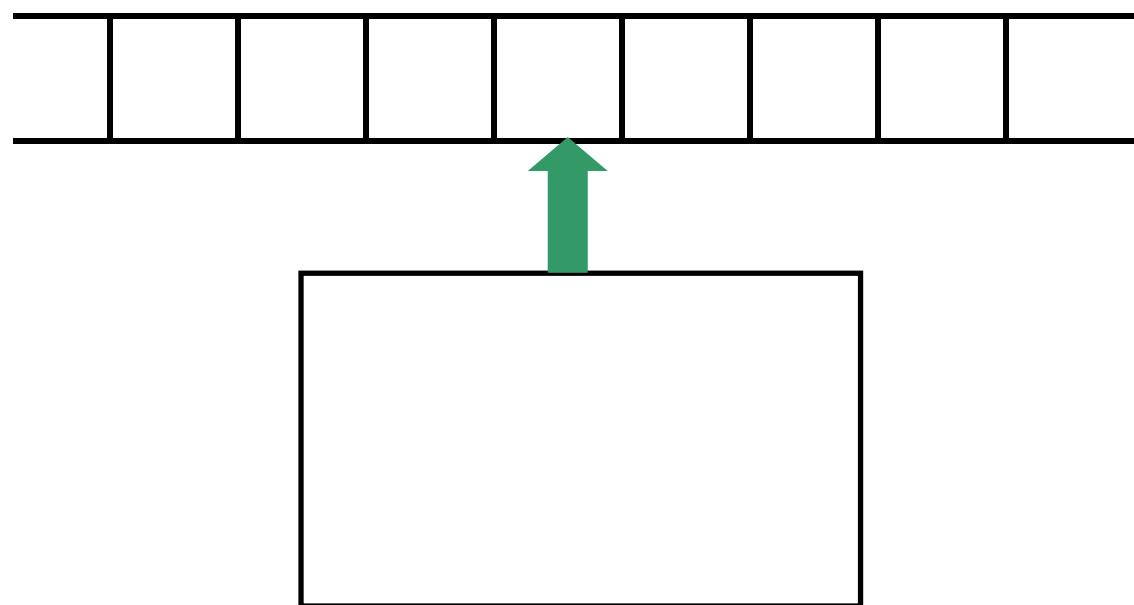
- Computing is normally done by writing certain symbols on paper. We may suppose this paper is divided into squares like a child's arithmetic book.
...
- I assume then that the computation is carried out on one-dimensional paper, i.e. on a tape divided into squares.

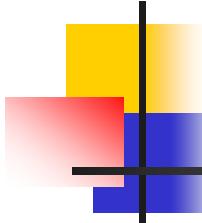




Turingによる計算の機械化(2)

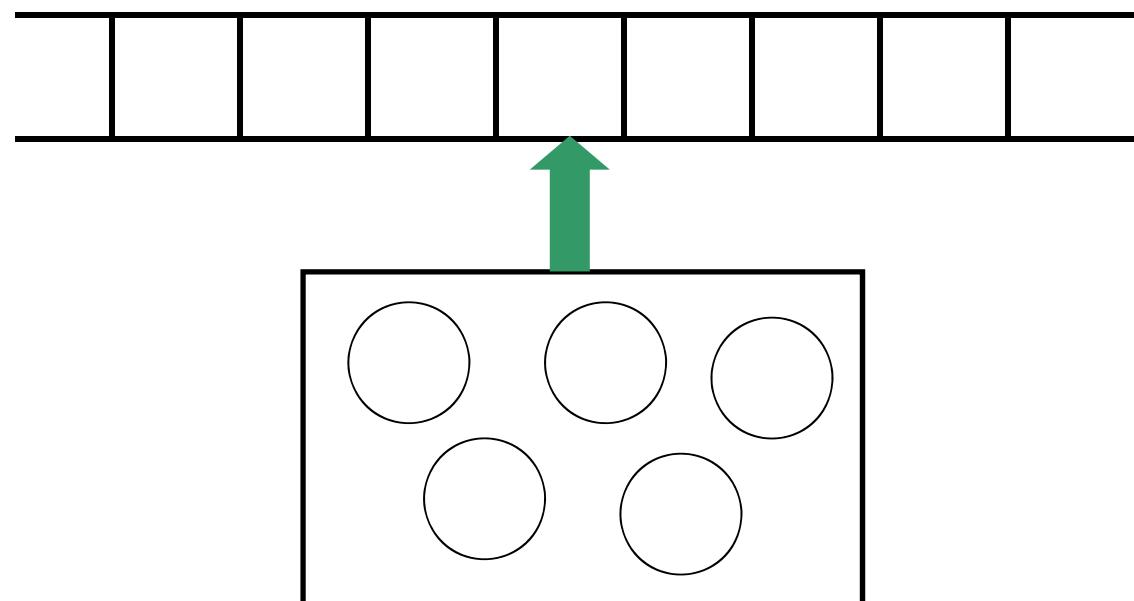
- The behaviour of the computer at any moment is determined by the symbols which he is observing and his “state of mind” at that moment.

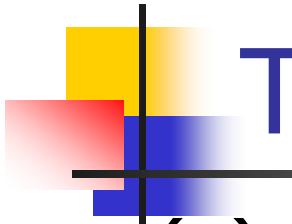




Turingによる計算の機械化(3)

- We will also suppose that the number of states of mind which need be taken into account is finite.





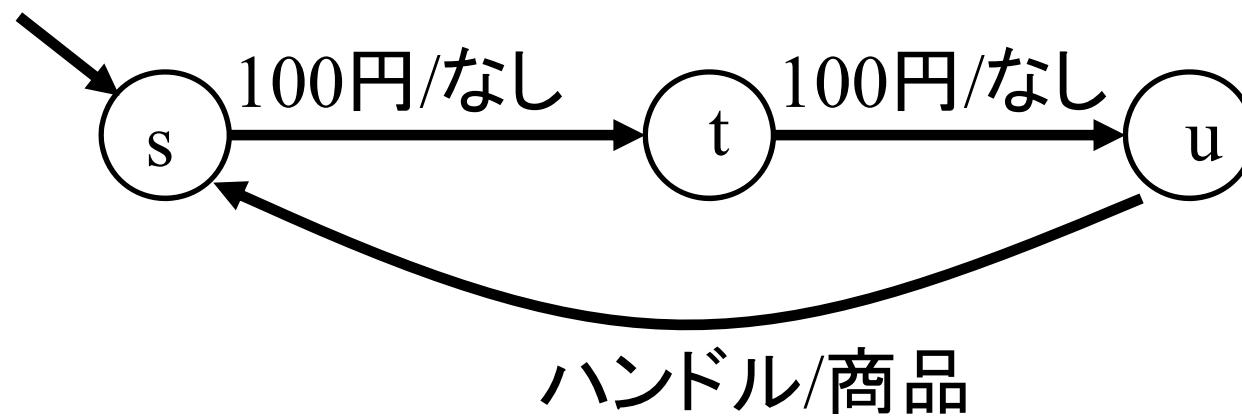
Turingによる計算の機械化(4)

- (a) Changes of the symbol on one of the observed squares.
 - (b) Changes of one of the squares observed to another square within L squares of one of the previously observed squares.
-
- A. A possible change (a) of symbol together with a possible change of state of mind.
 - B. A possible change (b) of observed squares, together with a possible change of state of mind.

有限状態機械

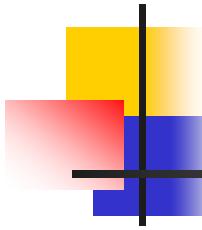
- 状態: 今流では「モード」

例 100円硬貨専用で200円商品の自動販売機



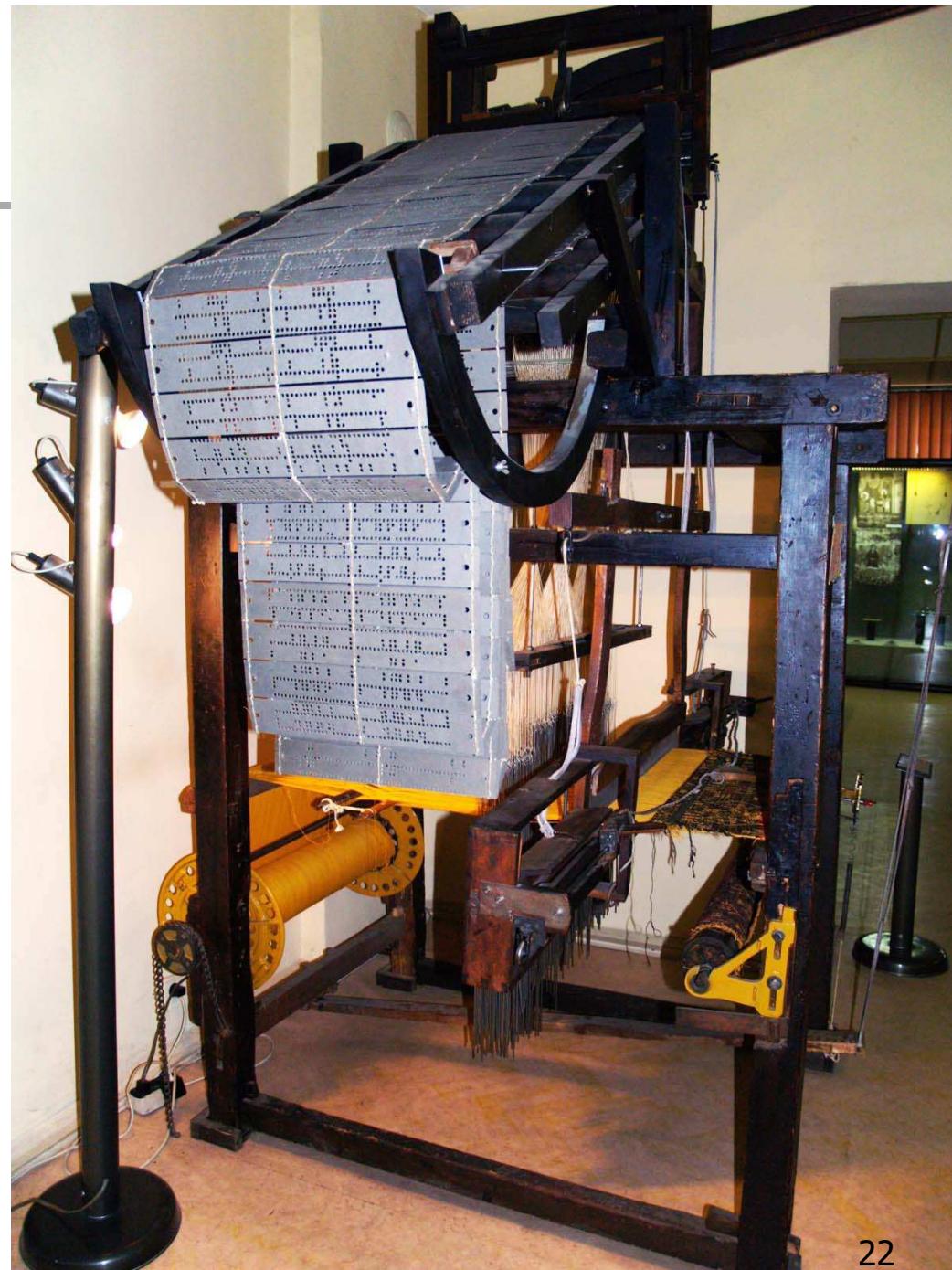
	100円	ハンドル
s	t/なし	s/なし
t	u/なし	t/なし
u	u/なし	s/商品

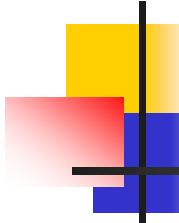




Jaccard織機

wikipediaより





テープに用いる記号

- 空白 B

記号 $a, b, \dots, 0, 1, \dots, +, *, \dots, \$, \dots$

- 簡単ため1進法を用いるときには

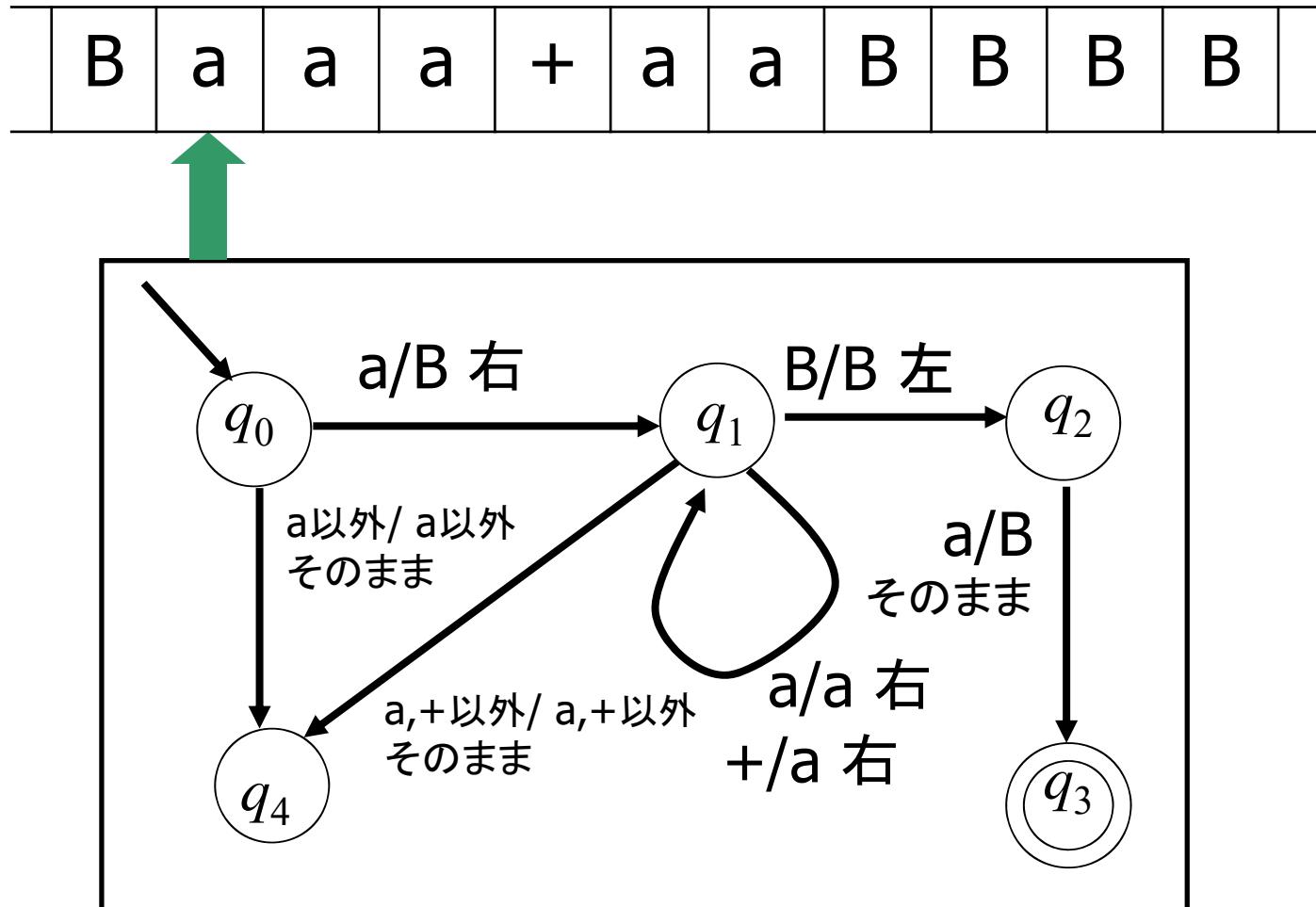
$0 \rightarrow a$

$1 \rightarrow aa$

$2 \rightarrow aaa$

\dots

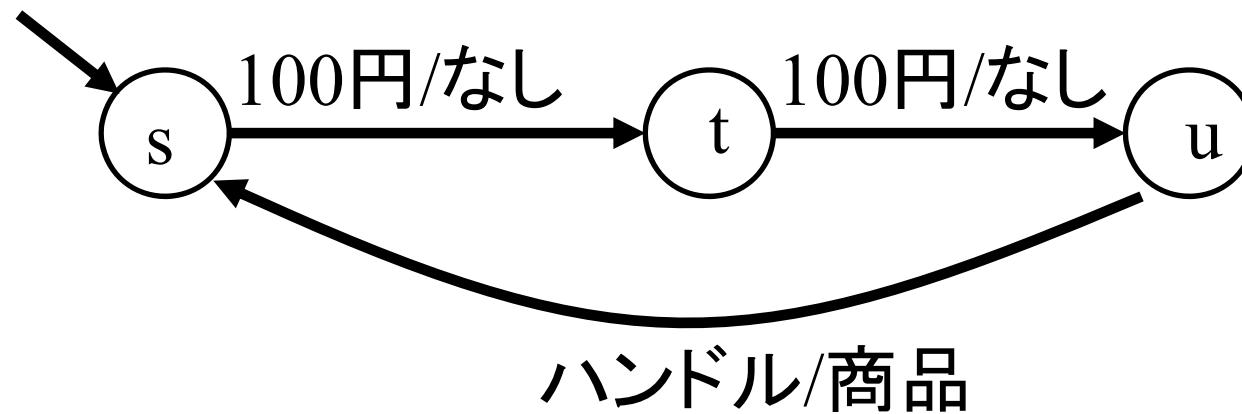
1進数の加算のためのTM



有限状態機械

- 状態: 今流では「モード」

例 100円硬貨専用で200円商品の自動販売機



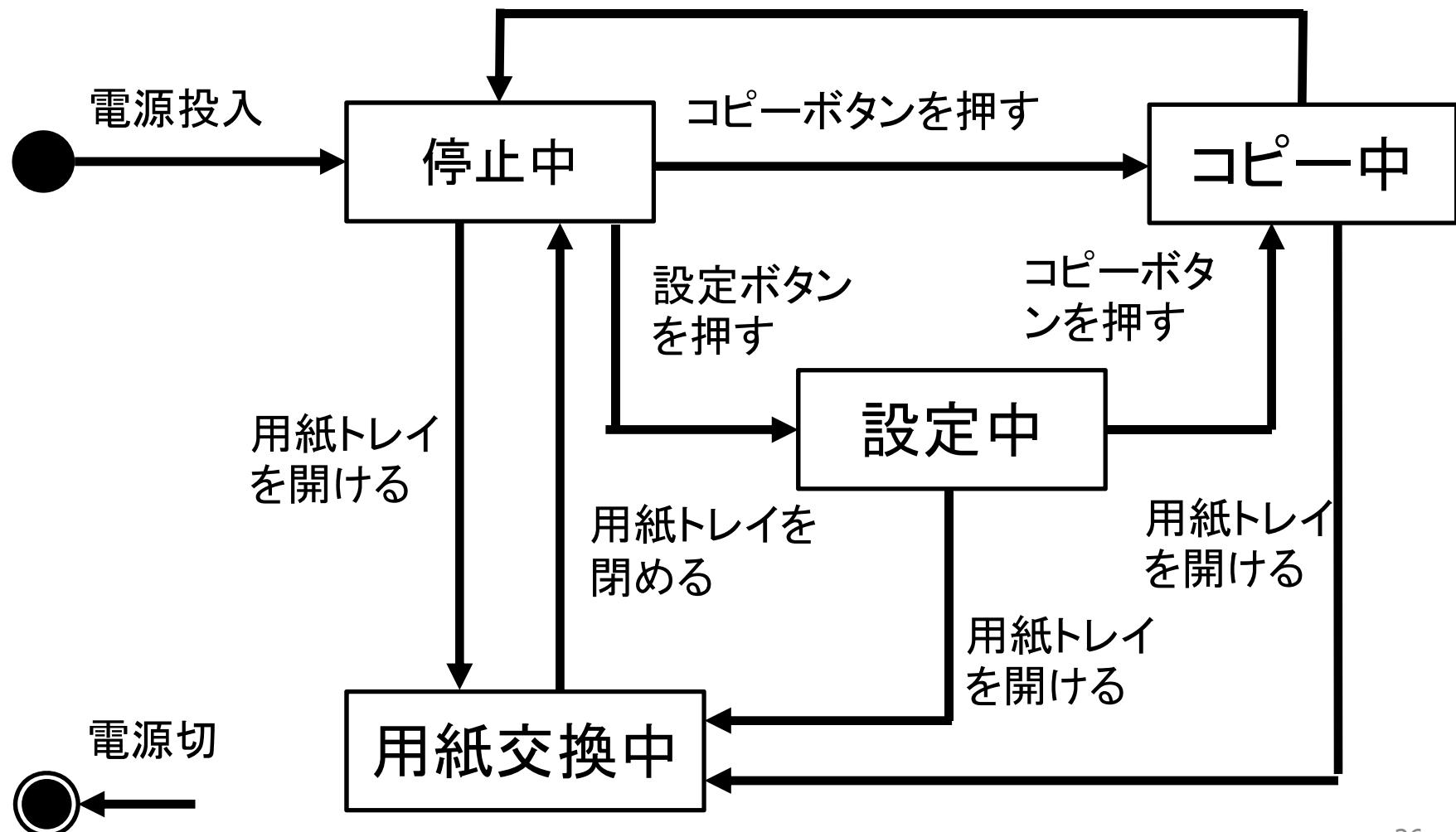
	100円	ハンドル
s	t/なし	s/なし
t	u/なし	t/なし
u	u/なし	s/商品

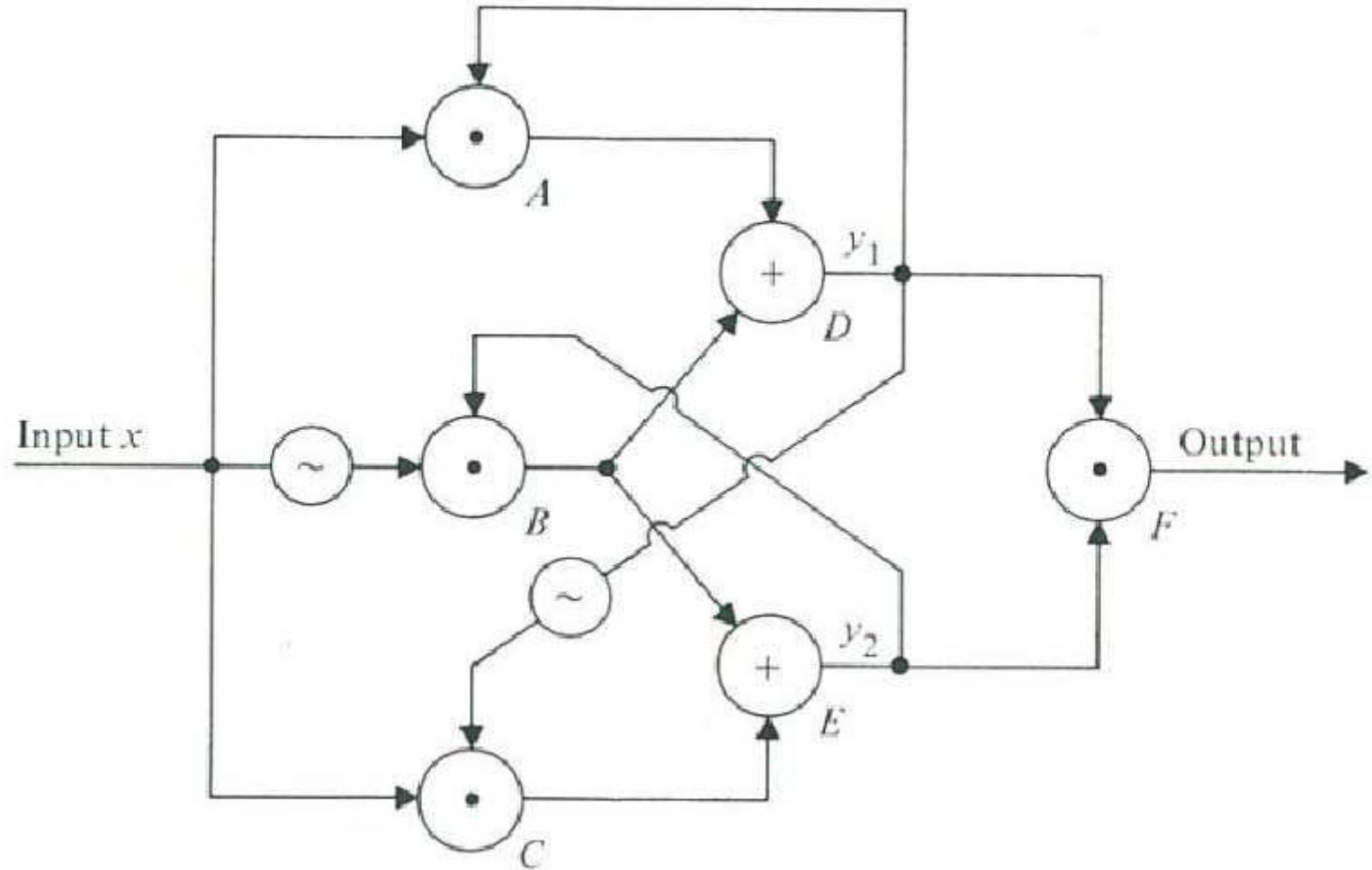


有限状態機械

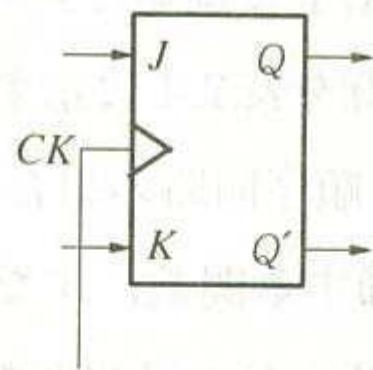
・ コピー機[石原・田中]

コピー終了を検知
停止ボタンを押す
用紙なしを検知





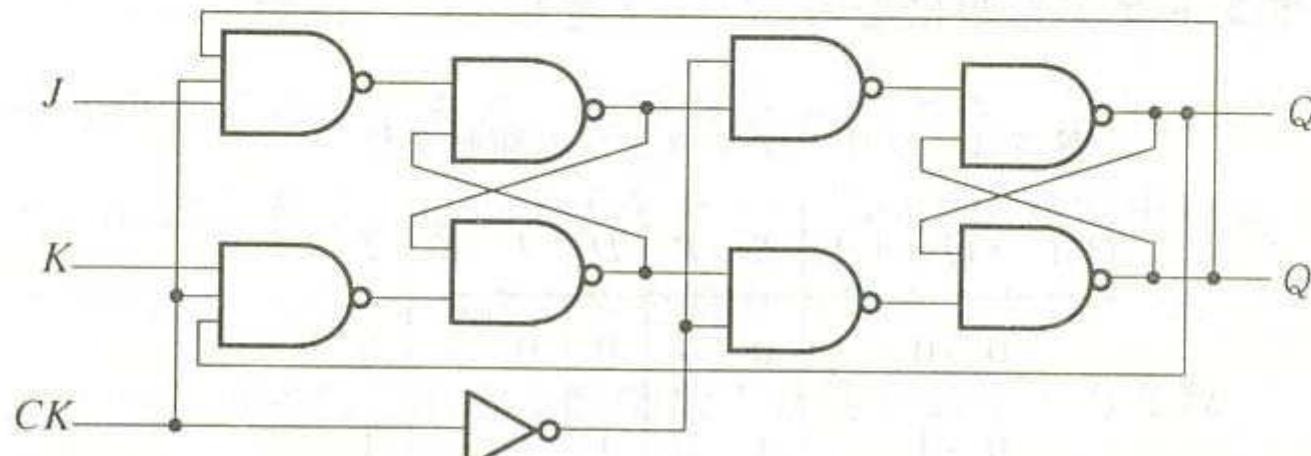
5.



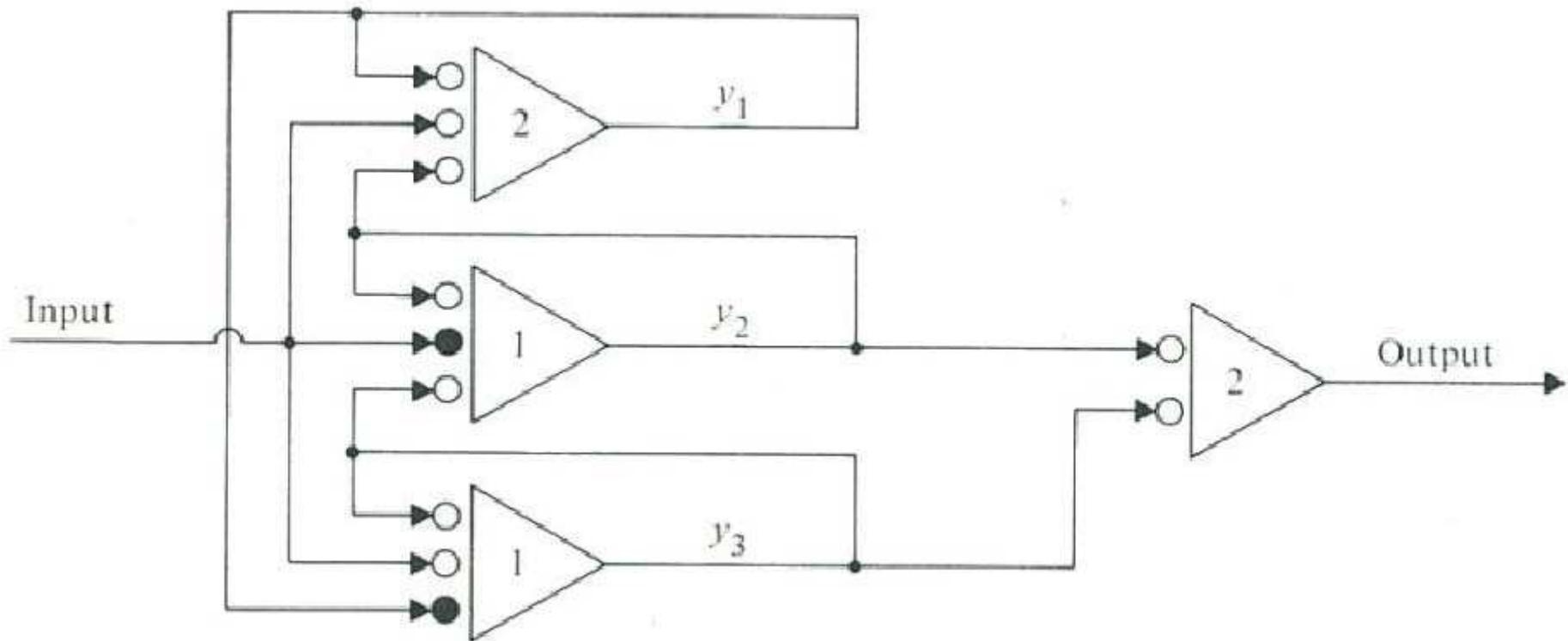
(a) 回路記号

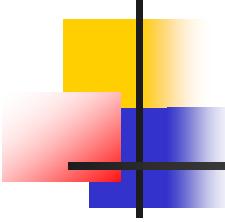
J	K	$Q(t+1)$
0	0	$Q(t)$
0	1	0
1	1	$Q(t)'$
1	0	1

(b) 動作表



(c) 回路構成

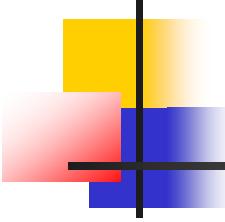




集合と関数による順序機械の表現(1)

- 以下のような構成要素からなる組 $M=(\Sigma, S, \delta, \lambda)$ をMealy型順序機械とよぶ。
 - Σ は入力アルファベット
 - Δ は出力アルファベット
 - S は空でない集合であり、その要素を状態とよぶ。
 - δ は $S \times \Sigma \rightarrow S$ なる関数
 - λ は $S \times \Sigma \rightarrow \Delta$ なる関数
 - δ と λ は右のような表で表現される

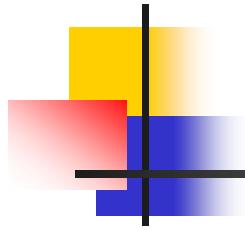
	a_1	\dots	a_n
q_0			
\dots			
q_m			



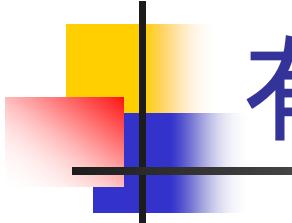
集合と関数による順序機械の表現(2)

- 以下のような構成要素からなる組 $M=(\Sigma, S, \delta, \lambda)$ をMoore型順序機械とよぶ。
 - Σ は入力アルファベット
 - Δ は出力アルファベット
 - S は空でない集合であり、その要素を状態とよぶ。
 - δ は $S \times \Sigma \rightarrow S$ なる関数
 - λ は $S \rightarrow \Delta$ なる関数
 - δ と λ は右のような表で表現される

	a_1	\dots	a_n	λ
q_0				
\dots				
q_m				



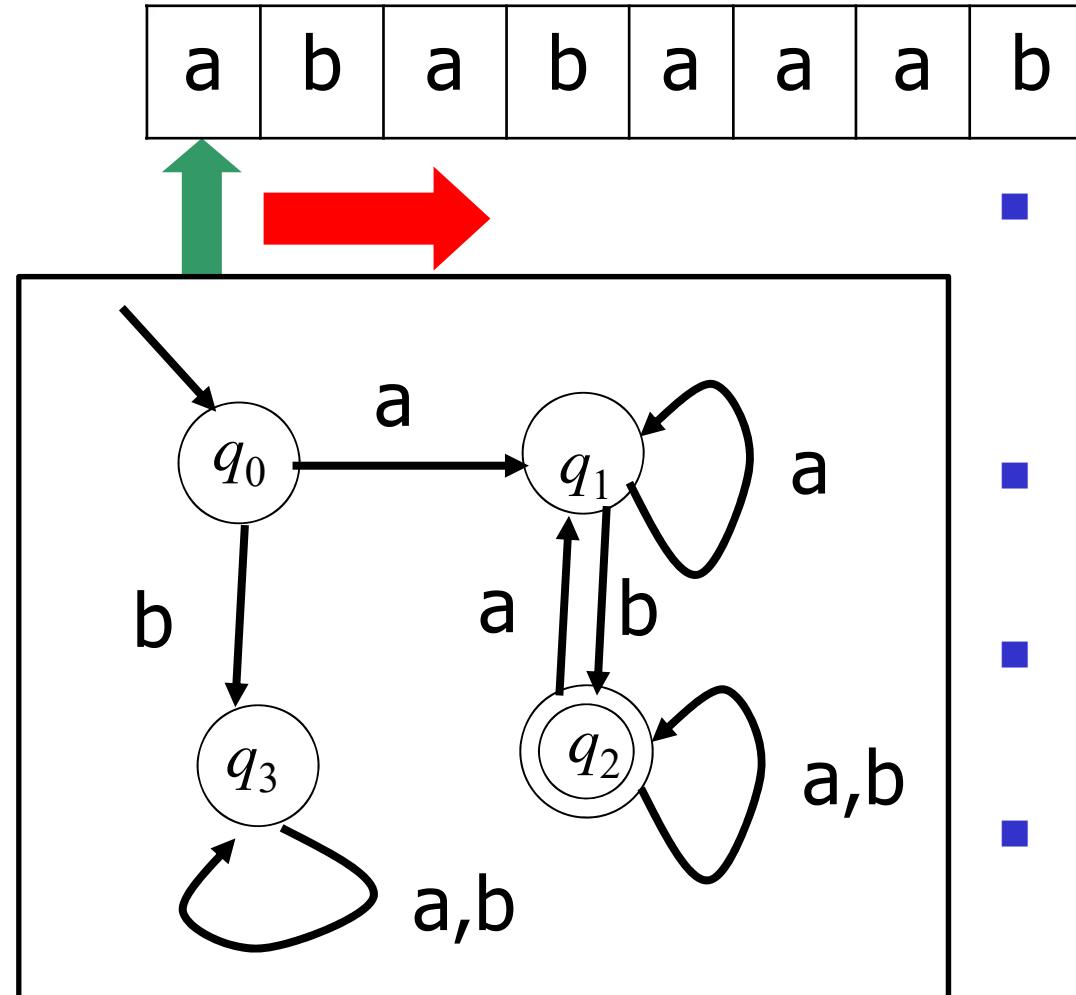
有限状態オートマトンと形式言語



有限状態オートマトン

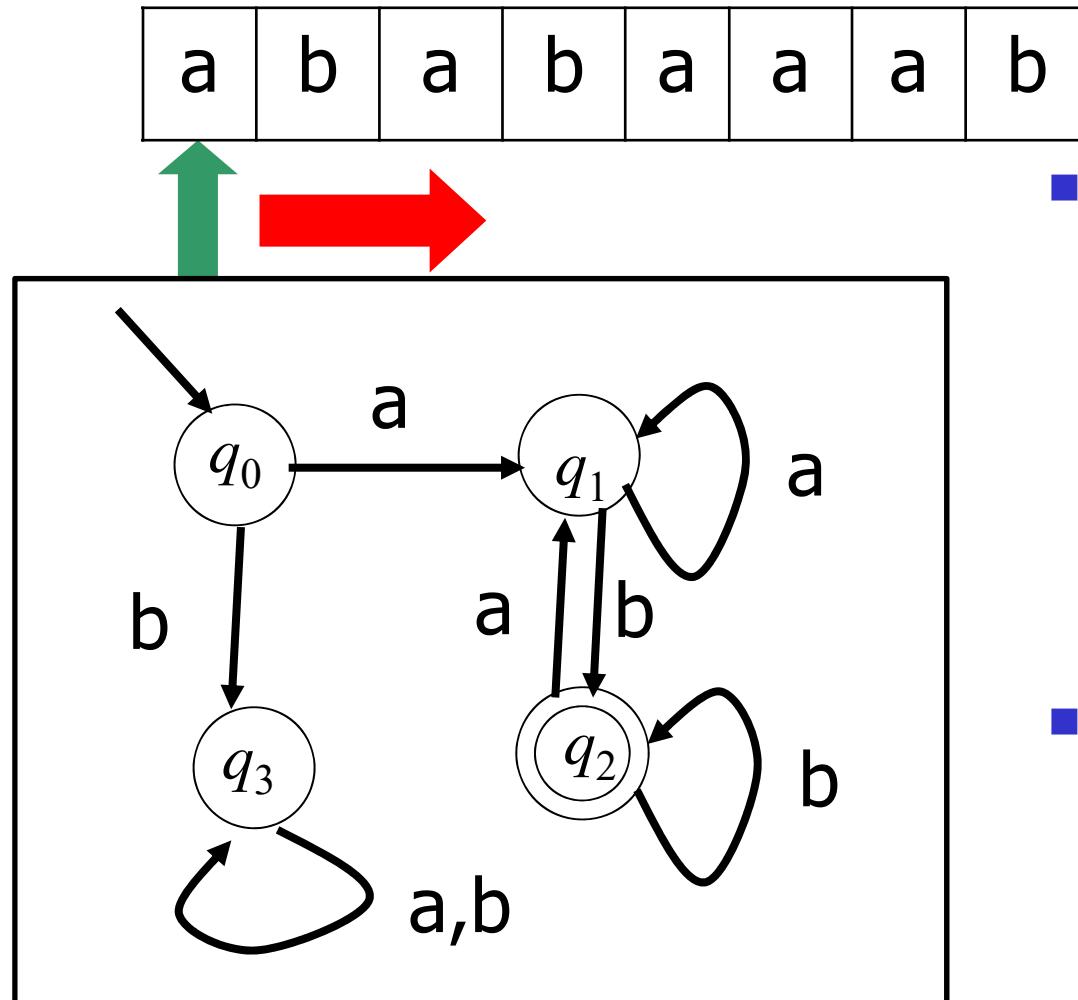
- ヘッドが一方向(右)にしか動かない
- テープは一方向に無限長
- 入力記号列は有限長(残りは空白で埋められる)
- 記号の書換えを行わない
- 入力記号列を読み終えたときに,
 - 終了状態にあれば, 入力記号列は“受理”
 - 終了状態になければ, 入力記号列は“不受理”

有限状態オートマトン



- テープは有限長で、記号ですべて埋められている
- ヘッドは必ず左端から右端へ動く。
- 記号を変更することはできない。
- 状態遷移は必ず初期状態から始まる
- 終了状態が決まっている(1個とは限らない)

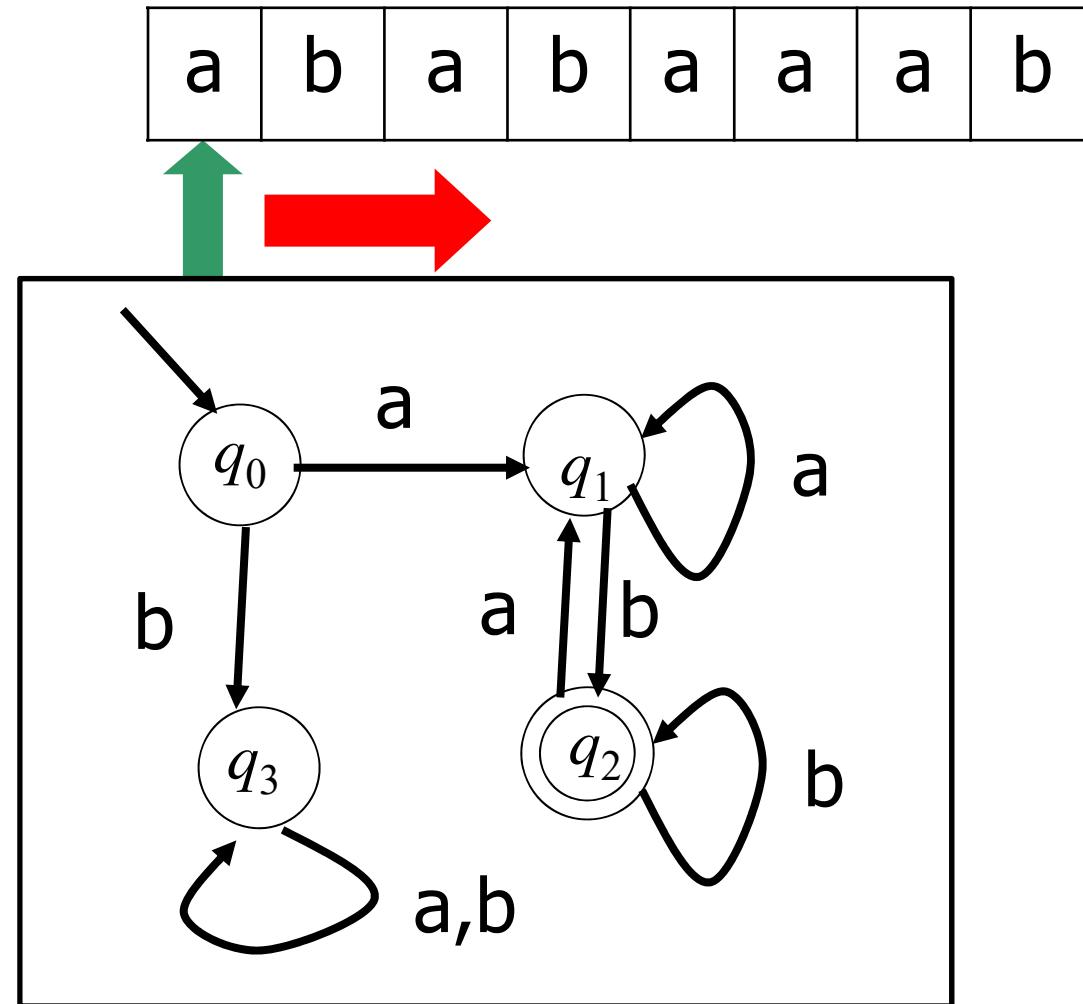
記号列の受理と言語



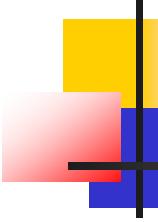
- テープ上の記号をすべて読み終えたときに、終了状態になっているとき、テープ上の記号列は**受理される**という。
- M によって受理される記号列全体は**言語**である。

$$L(M) = \{aab, abb, aaab, aabb, abab, \dots\}$$

有限状態オートマトンの表現



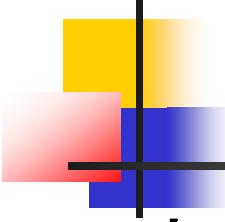
	F	a	b
q_0		q_1	q_3
q_1		q_1	q_2
q_2	○	q_1	q_2
q_3		q_3	q_3



集合と関数による有限状態オートマトンの表現

- 以下のような構成要素からなる組 $M=(\Sigma, S, \delta, s_0, F)$ を(決定性)有限状態オートマトンとよぶ.
 - Σ はアルファベット
 - S は空でない集合であり,
その要素を状態とよぶ.
 - δ は $S \times \Sigma \rightarrow S$ なる関数
 - δ は右のような表で表現される
 - $s_0 \in S$ は特定の状態であり, 初期状態とよぶ.
 - $F \subset S$ は特定の状態の集合であり, その要素を終了状態, もしくは受理状態とよぶ.

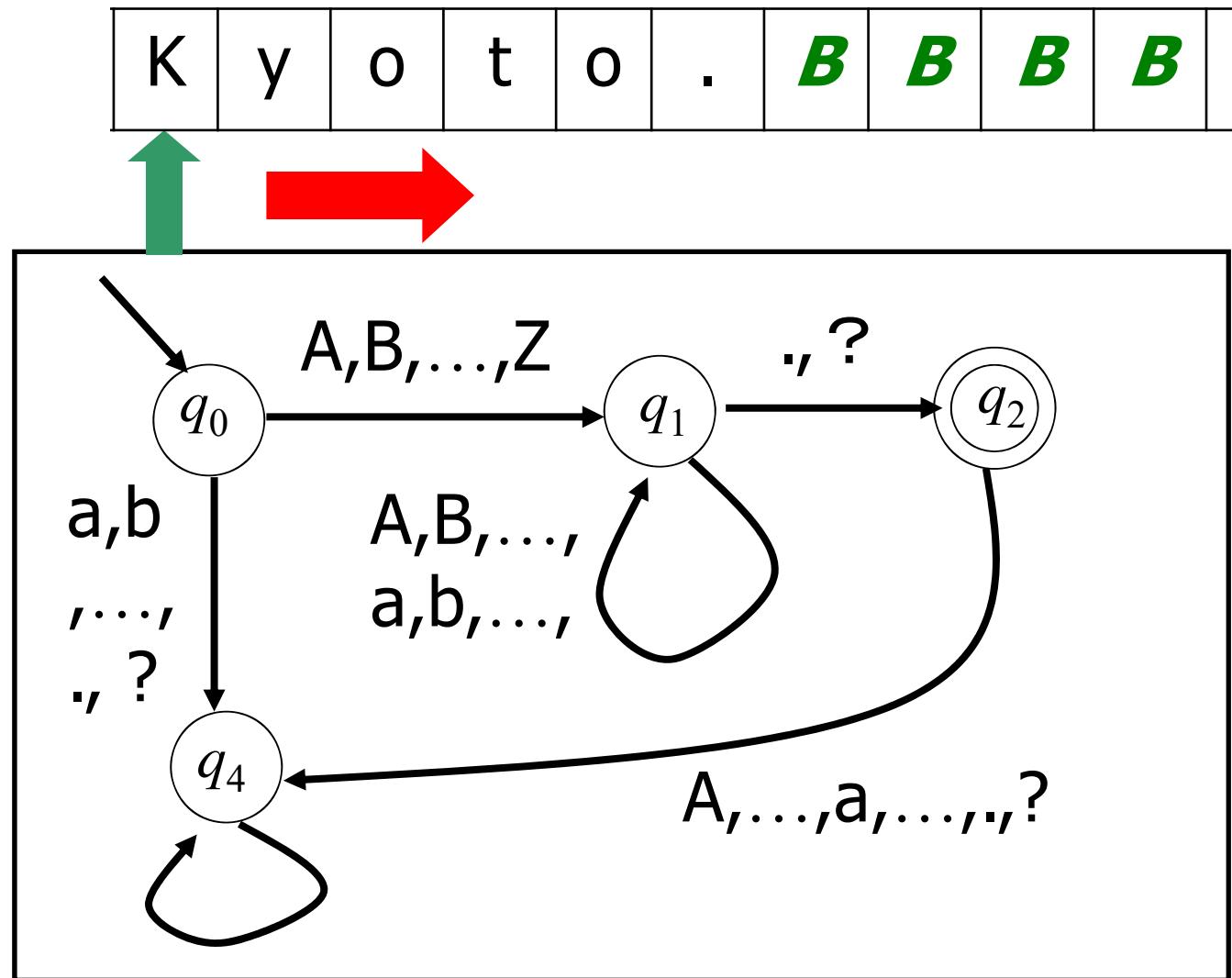
	F	a_1	\dots	a_n
q_0				
\dots				
q_m				



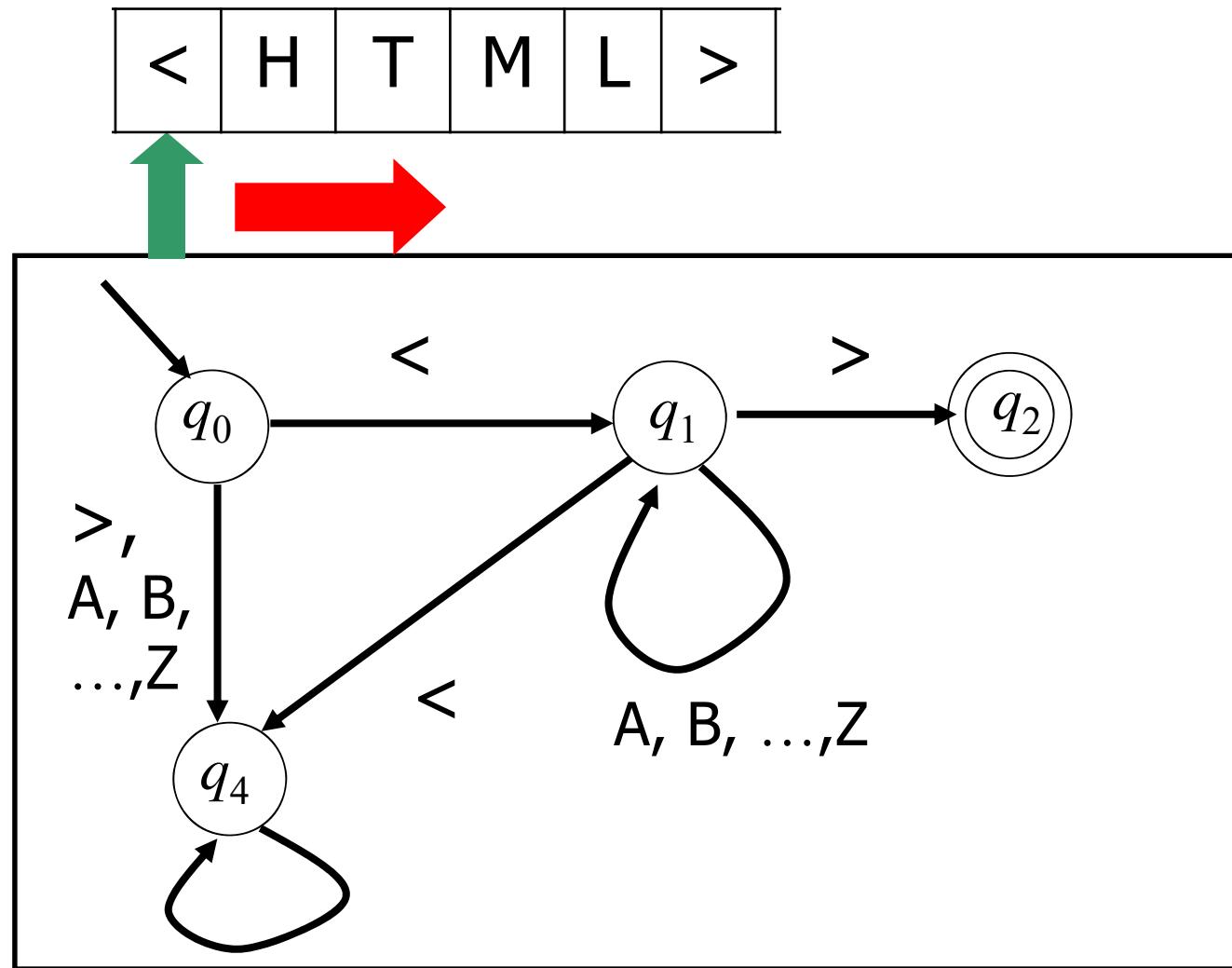
正則言語

- ある言語 L について $L = L(M)$ であるとき, 有限状態オートマトン M は L を受理するという.
- 有限状態オートマトンで受理される言語を正則言語(regular language)という.

有限状態オートマトン(例)

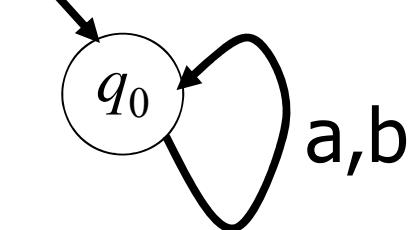


有限状態オートマトン(例)

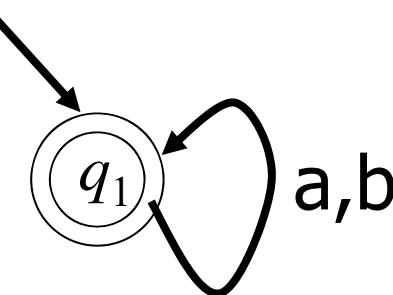


状態数1のオートマトン

M_0



M_1



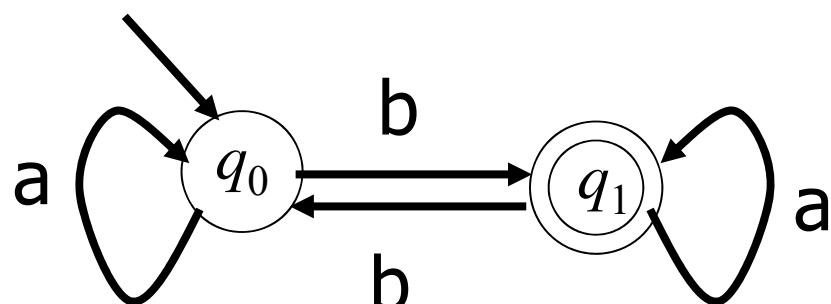
	F	a	b
q_0		q_0	q_0

$$L(M_0) = \emptyset$$

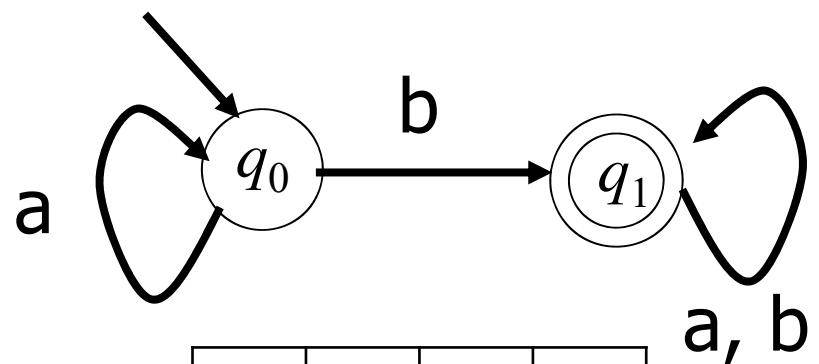
	F	a	b
q_0	○	q_0	q_0

$$L(M_1) = \Sigma^*$$

状態数2のオートマトン



	F	a	b
q_0		q_0	q_1
q_1	○	q_1	q_0



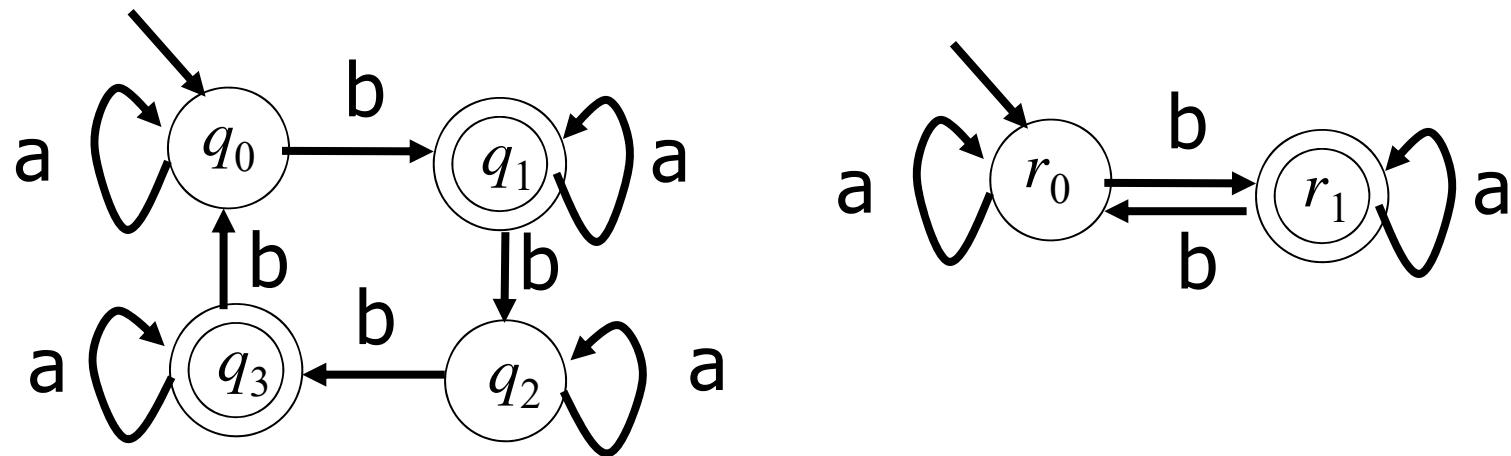
	F	a	b
q_0		q_0	q_1
q_1	○	q_1	q_1



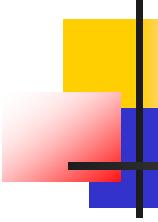
有限状態オートマトンの等価性

等価なオートマトン

- 下の2つの有限状態オートマトンが受理する言語は一致する



- 有限状態オートマトン M_1 と M_2 に対して $L(M_1) = L(M_2)$ であるとき, M_1 と M_2 は等価(equivalent)であるといい, $M_1 \equiv M_2$ と表す.



等価なオートマトンに関する問題

問題1: 有限状態オートマトン M_1 と M_2 に対して $M_1 \equiv M_2$ であるかどうかをどのようにして判定するのか？

- $M_1 \equiv M_2$ を判定する“アルゴリズム”を与えよ.

問題2: 有限状態オートマトン M に対して $M \equiv M'$ を満たす M' で、状態数が最小のものを求めよ.

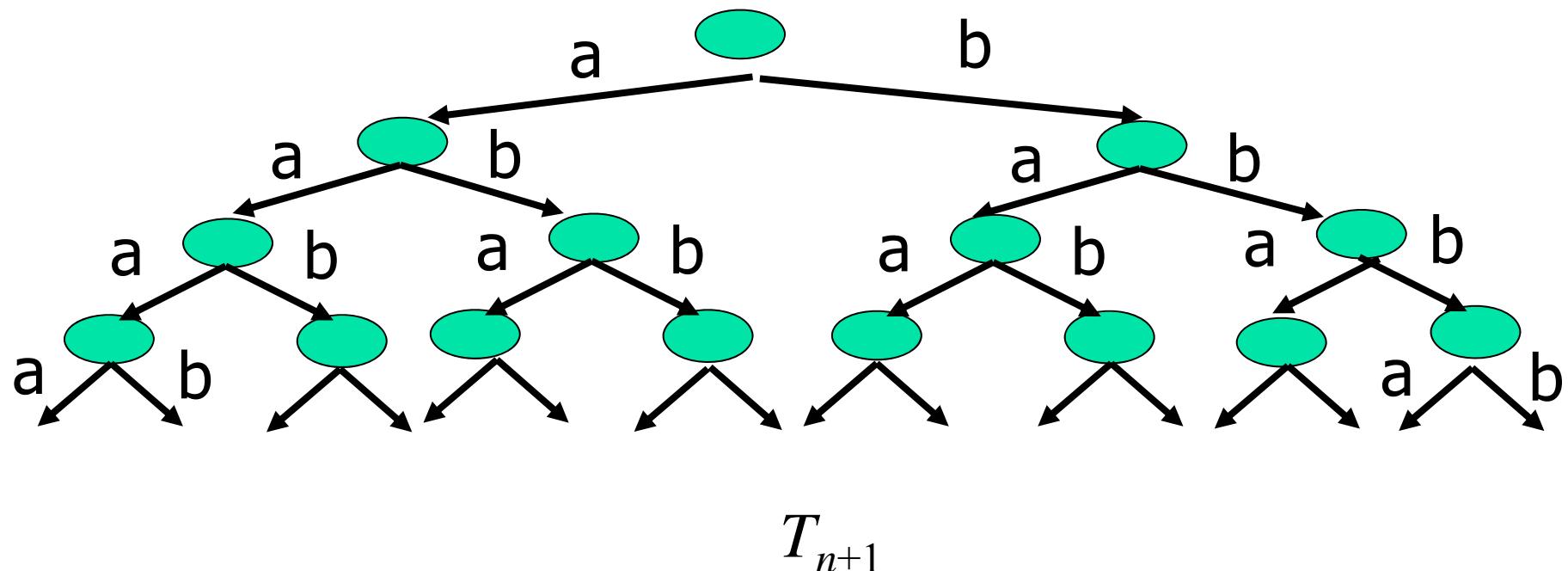
- 数学的にはこのような M' が存在することはわかる
- このような M' は唯一なのか？複数個あるのか？
- M' を求める“アルゴリズム”を与えよ.

オートマトンの等価性判定の準備

- 長さ $n+1$ 以下のすべての記号列の集合 $\Sigma^*|_{\leq n+1}$

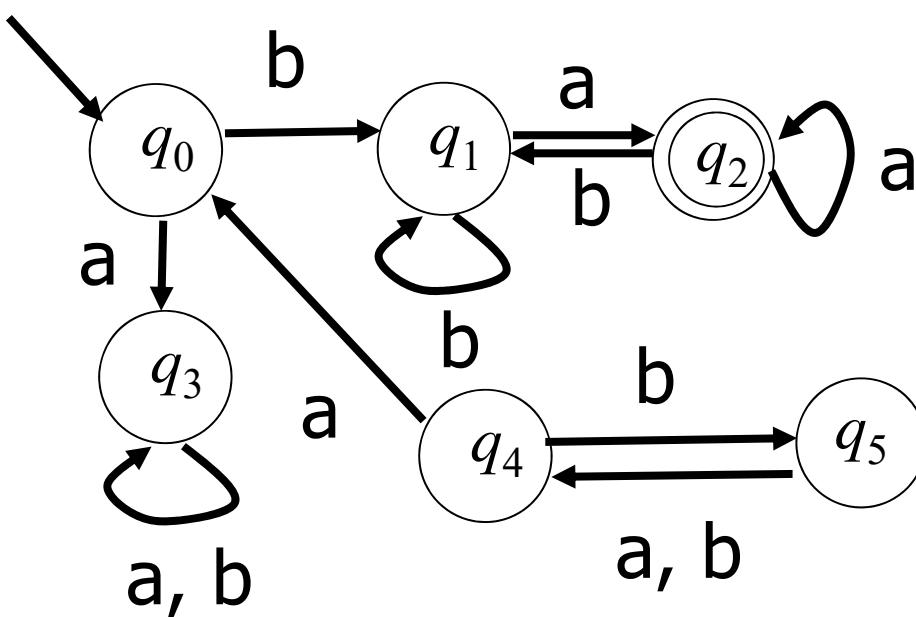
例 $\Sigma = \{a, b\}$

$$\Sigma^*|_{\leq n+1} = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, \dots, a^{n+1}, \dots, b^{n+1}\}$$



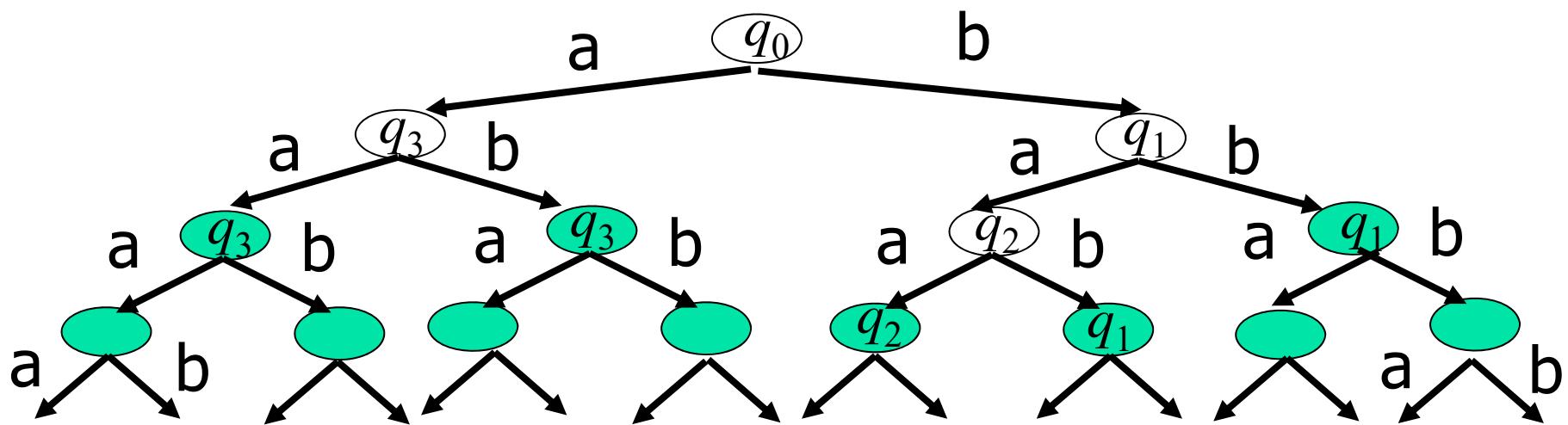
到達不可能な状態

- 下図の有限状態オートマトン M において、状態 q_4, q_5 は初期状態からどのような状態遷移によっても到達できない
 - 到達可能・不可能の数学的定義は後で
- M から q_4, q_5 を除いて得られる有限状態オートマトン M' については $M \equiv M'$



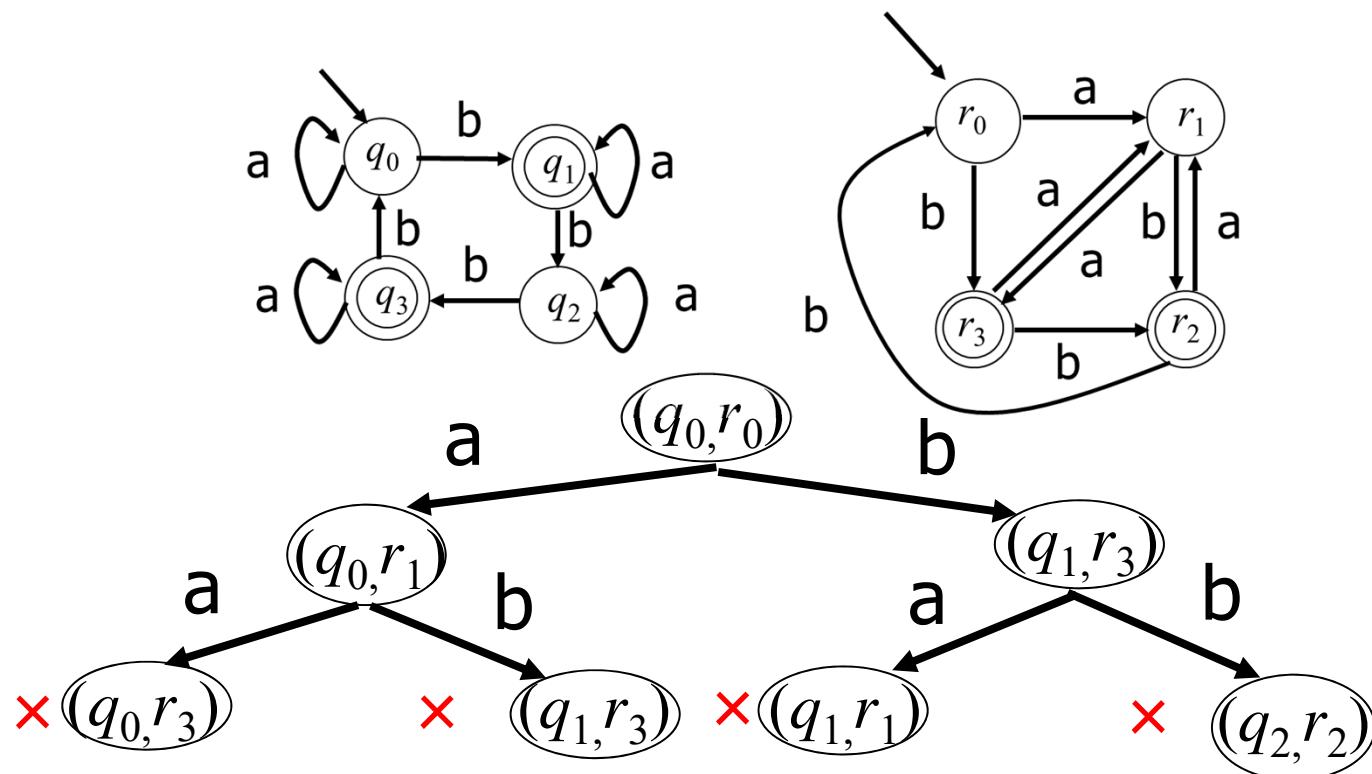
到達不可能な状態の検出

- T_{n+1} の根から順に節点(ノード)に M の状態をラベル付けする
 - $n = M$ の状態数
 - 同じ状態が出てきたら、それ以上操作を続ける必要はない



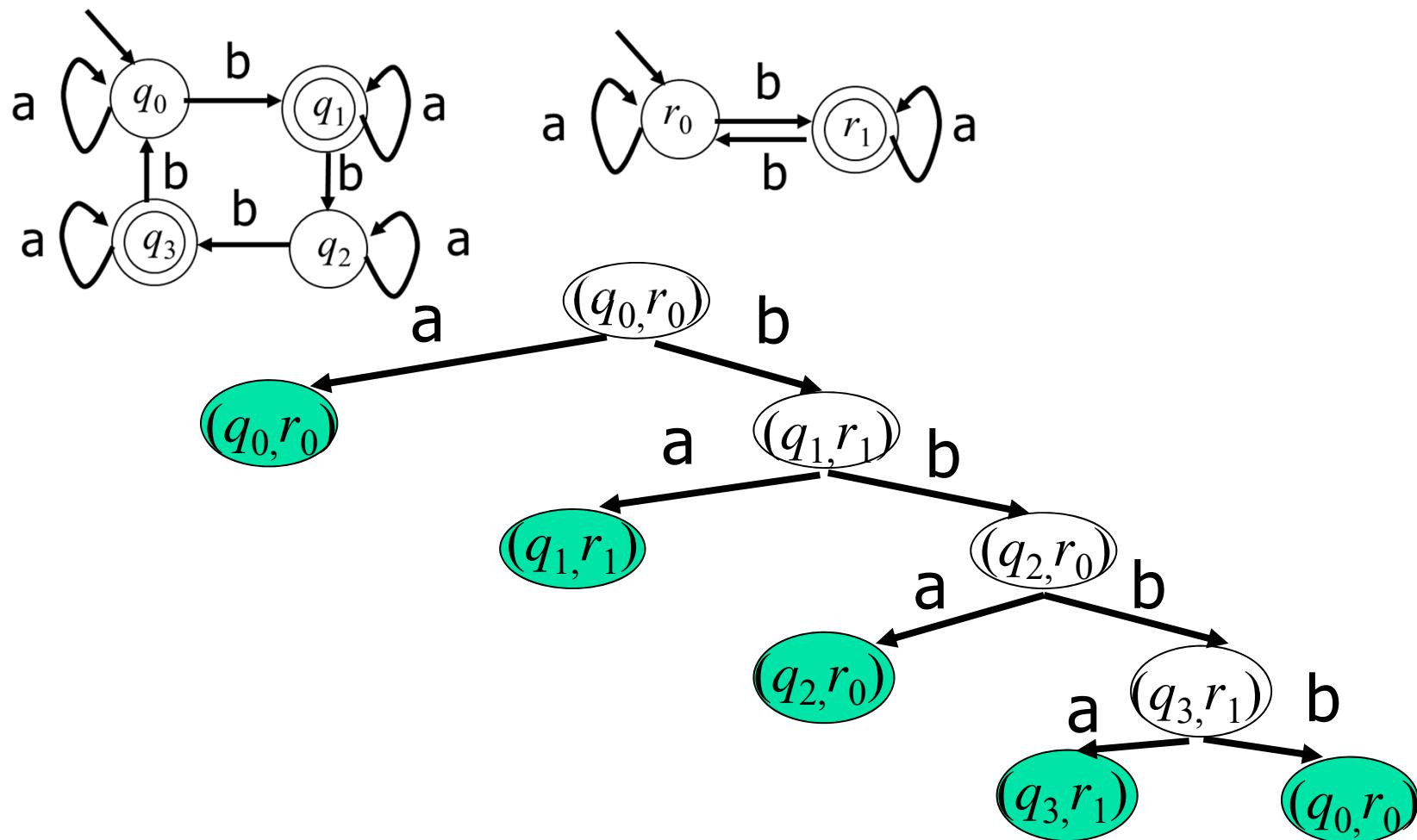
等価性判定(1)

- M_1 の状態数 n_1 , M_2 の状態数 n_2 , $n = \max(n_1, n_2)$
- T_{n+1} の節点(ノード)に M_1 の状態と M_2 の状態の組をラベル付けする
- 終了状態と非終了状態の組が出てきたら終了



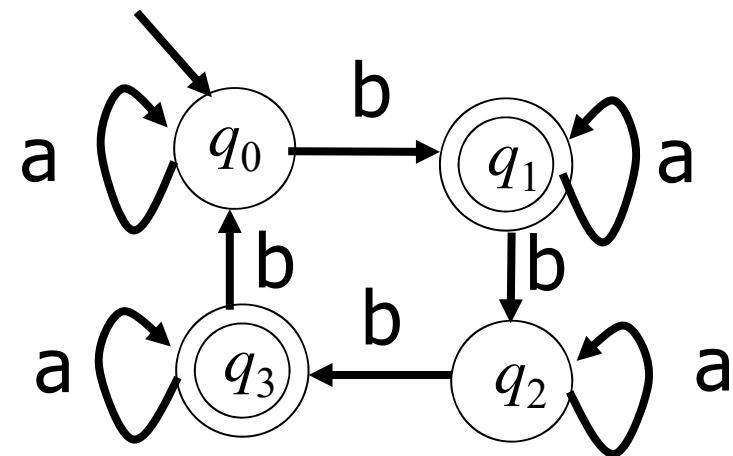
等価性判定(2)

- 同じ状態の組が出てきたら、それ以上操作を続ける必要はない

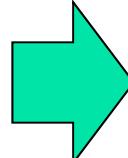


状態数最小化(step 1)

- 状態を終了状態と非終了状態にだけ分類して、状態遷移表を書き換え
- 各行のパターンを分類



	F	a	b	
q_0	○	q_0	q_1	
q_1		q_1	q_2	
q_2	○	q_2	q_3	
q_3		q_3	q_0	



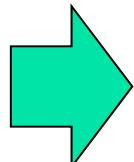
	F	a	b	
q_0	1	1	0	A
q_1	0	0	1	B
q_2	1	1	0	A
q_3	0	0	1	B

状態数最小化(step k)

- 各行のパターンを用いて、遷移表を書き換えてから各行のパターンを再分類

	F	a	b
q_0	○	q_0	q_1
q_1		q_1	q_2
q_2	○	q_2	q_3
q_3		q_3	q_0

	F	a	b	
q_0	1	1	0	A
q_1	0	0	1	B
q_2	1	1	0	A
q_3	0	0	1	B



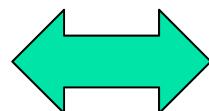
	F	a	b	
q_0	1	A	B	A'
q_1	0	B	A	B'
q_2	1	1	0	A'
q_3	0	0	1	B'

状態数最小化(step k)

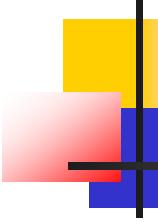
- 行の分類がもとの遷移表と変化がなければ終了。
変化があれば、前頁へ戻る

	F	a	b
q_0	○	q_0	q_1
q_1		q_1	q_2
q_2	○	q_2	q_3
q_3		q_3	q_0

	F	a	b	
q_0	1	1	0	A
q_1	0	0	1	B
q_2	1	1	0	A
q_3	0	0	1	B



	F	a	b	
q_0	1	A	B	A'
q_1	0	B	A	B'
q_2	1	1	0	A'
q_3	0	0	1	B'



複雑な例(2)

	F	a	b
q_0		q_1	q_2
q_1		q_4	q_7
q_2	○	q_2	q_0
q_3	○	q_2	q_1
q_4		q_2	q_6
q_5		q_4	q_7
q_6	○	q_5	q_1
q_7		q_5	q_6

複雑な例

	F	a	b	
q_0		q_1	q_2	
q_1		q_4	q_7	
q_2	○	q_2	q_0	
q_3	○	q_2	q_1	
q_4		q_2	q_6	
q_5		q_4	q_7	
q_6	○	q_5	q_1	
q_7		q_5	q_6	

	F	a	b	
q_0	0	0	1	A
q_1	0	0	0	B
q_2	1	1	0	C
q_3	○	q_2	q_1	
q_4	0	1	1	D
q_5	0	0	0	B
q_6	1	0	0	E
q_7	0	0	1	A

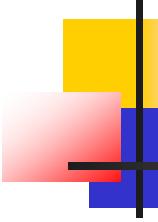
何気ないオートマトンの最小化が到達不能状態を消し去った

複雑な例(2)

	F	a	b	
q_0		q_1	q_2	
q_1		q_4	q_7	
q_2	○	q_2	q_0	
q_3	○	q_2	q_1	
q_4		q_2	q_6	
q_5		q_4	q_7	
q_6	○	q_5	q_1	
q_7		q_5	q_6	

	F	a	b	
q_0	0	0	1	A
q_1	0	0	0	B
q_2	1	1	0	C
q_3	○	q_2	q_1	
q_4	0	1	1	D
q_5	0	0	0	B
q_6	1	0	0	E
q_7	0	0	1	A

	F	a	b	
q_0	0	B	C	A'
q_1	0	D	A	B'
q_2	1	C	A	C'
q_3	○	q_2	q_1	
q_4	0	C	E	D'
q_5	0	D	A	B'
q_6	1	B	B	E'
q_7	0	B	E	F'

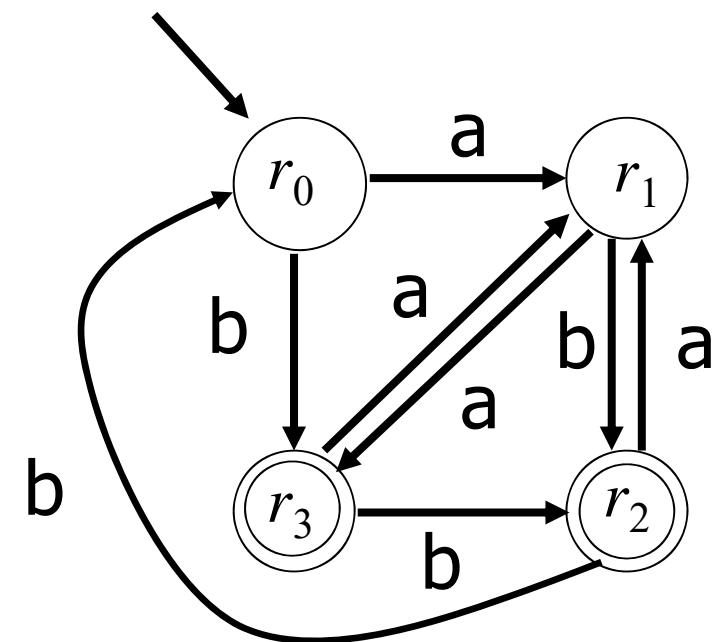
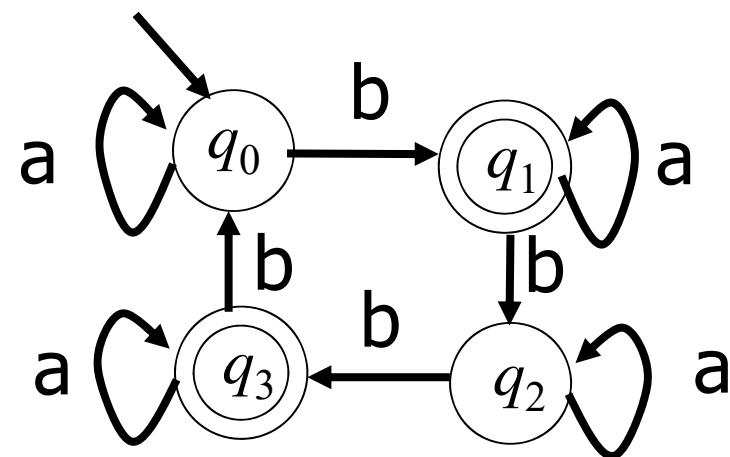


複雑な例(2)

	F	a	b
q_0		q_1	q_2
q_1		q_4	q_7
q_2	○	q_2	q_0
q_3	○	q_2	q_1
q_4		q_2	q_6
q_5		q_4	q_7
q_6	○	q_5	q_1
q_7		q_5	q_6

	F	a	b	
q_0	0	B	C	A'
q_1	0	D	A	B'
q_2	1	C	A	C'
q_3	○	q_2	q_1	
q_4	0	C	E	D'
q_5	0	D	A	B'
q_6	1	B	B	E'
q_7	0	B	E	F'

	F	a	b	
q_0	0	B'	C'	A''
q_1	0	D'	F'	B''
q_2	1	C'	A'	C''
q_3	○	q_2	q_1	
q_4	0	C'	E'	D''
q_5	0	D'	F'	B''
q_6	1	B'	B'	E''
q_7	0	B'	E'	"





正則表現



計算機環境での正則表現

- もともとは数学者 S. Kleene が集合(形式言語)の表記方法として考案
- LinuxなどUNIX系のshellやWindowsのコマンドラインに正則表現を導入

```
$ ls *.c
```

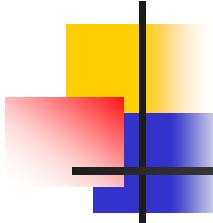
```
$ ls [abc]*.c
```

- Perlなどのプログラミング言語やスクリプト言語やでも正則表現を導入

```
/ab. cd/
```

```
/ab*cd/
```

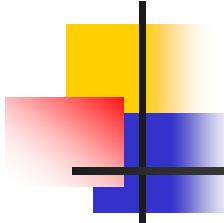
- パターンと呼ばれることも多い
- “文法”は言語やシステムによってまちまち



Σ^* 上の正則表現

Σ^* の部分集合(形式言語)を表すための式(expression)

1. \emptyset (太字)は空集合 $\{\}$ を表す(Σ 上の)正則表現である
2. ϵ (太字)は集合 $\{\epsilon\}$ を表す正則表現である
3. $c \in \Sigma$ に対して c は集合 $\{c\}$ を表す正則表現である
4. R, S を Σ^* の部分集合 R, S を表す正則表現としたとき
 - $(R + S)$ は集合 $R \cup S$ を表す正則表現である
 - (RS) は集合 RS を表す正則表現である
 - (R^*) は集合 R^* を表す正則表現である
5. 上の1から3から開始して, 4を**有限回適用して得られる式**だけが正則表現である.



形式言語のKleene閉包

- 形式言語 L に対して次のように定義される集合 L^* を L の Kleene閉包もしくは単に閉包という

$$L^0 = \{\varepsilon\} \text{ (空集合ではない!)}$$

$$L^n = L \quad L^n = \{ uv \mid u \in L, v \in L^{n-1} \} \quad (n \geq 1)$$

$$L^* = \{\varepsilon\} \cup L \cup L^2 \cup L^3 \cup \dots \stackrel{n=1}{=} \cup L^n$$

Example

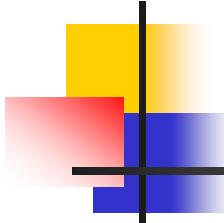
$L = \{aa, ab\}$ とすると

$$L^2 = \{aaaa, aaab, abaa, abab\}$$

$$L^3 = \{aaaaaaaa, aaaaab, aaabaa, aaabab, abaaaa, \dots\}$$

...

$$L^* = \{\varepsilon, aa, ab, aaaa, aaab, abaa, abab, aaaaaa, \dots\}$$



正則表現が表す言語

- 正則表現 R が表す言語を $L(R)$ とする

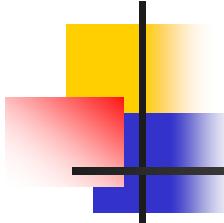
Examples

$$L(\mathbf{ab}) = \{\mathbf{a}, \mathbf{b}\}$$

$$L(\mathbf{a}(\Sigma^*)\mathbf{b}) = \{w \mid w = \mathbf{a}u\mathbf{b} \text{かつ } u \in \Sigma^*\}$$

$$\begin{aligned} L(\mathbf{a}(\mathbf{ab})^*\mathbf{b}) &= \{w \mid w = \mathbf{a}u\mathbf{b} \text{かつ } u \in \{\mathbf{ab}\}^*\} \\ &= \{\mathbf{ab}, \mathbf{aabb}, \mathbf{aababb}, \dots\} \end{aligned}$$

$$\begin{aligned} L(\mathbf{a}(\mathbf{aa}+\mathbf{ab})^*\mathbf{b}) &= \{w \mid w = \mathbf{a}u\mathbf{b} \text{かつ } u \in \{\{\mathbf{aa}\} \cup \{\mathbf{ab}\}\}^*\} \\ &= \{\mathbf{aaab}, \mathbf{aabb}, \mathbf{aaaaab}, \mathbf{aaaabb}, \mathbf{aababaab}, \\ &\quad \mathbf{aababb}, \mathbf{aaaaaaaaab}, \dots\} \end{aligned}$$



正則表現の等価性

- 正則表現 R, S について $L(R)=L(S)$ のとき, R と S は等価であるといい, $R = S$

Example

$$(a+b)^* = ((a^*)(b^*))^*$$

公式集

$$\emptyset^* = \varepsilon \quad \varepsilon R = R\varepsilon = R \quad R + R = R \quad R + S = S + R$$

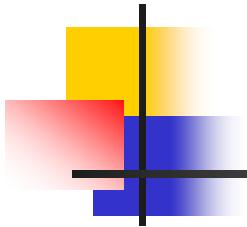
$$(R + S)T = ST + RT \quad T(R + S) = TS + TR$$

$$(R^*)^* = R^* \quad (R^*) (R^*)^* = R^* \quad R^* R + \varepsilon = R^*$$

$$(R^*S^*)^* = (R + S)^* \quad (R^*S)^* R^* = (R + S)^*$$

$$(R^*S)^* + (S^*R) = (R + S)^* \quad (R + S)^* S + \varepsilon = (R^* S)^*$$

$$R^*S + S = R^*S \quad RS^* + R = RS^*$$

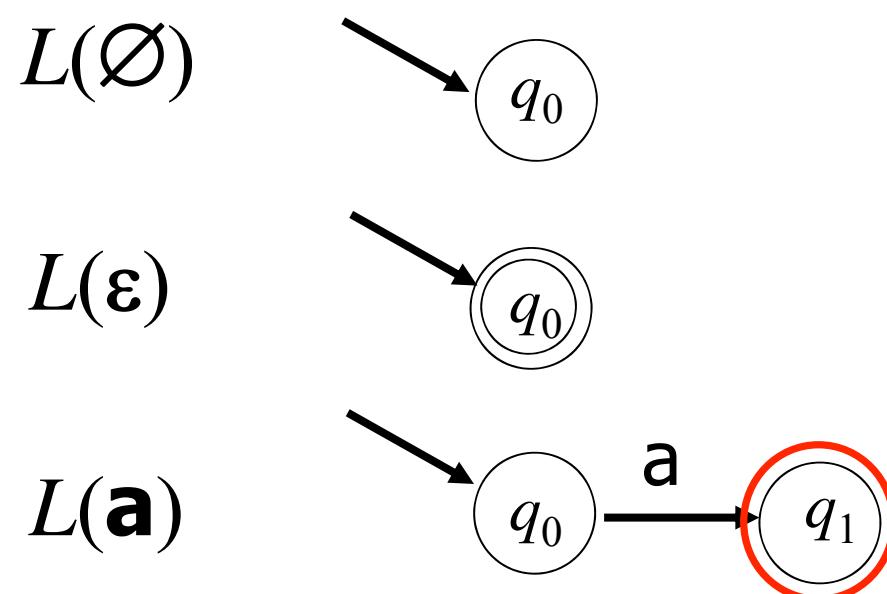


正則表現から有限状態オートマトン への変換

基本的なアイデア(1)

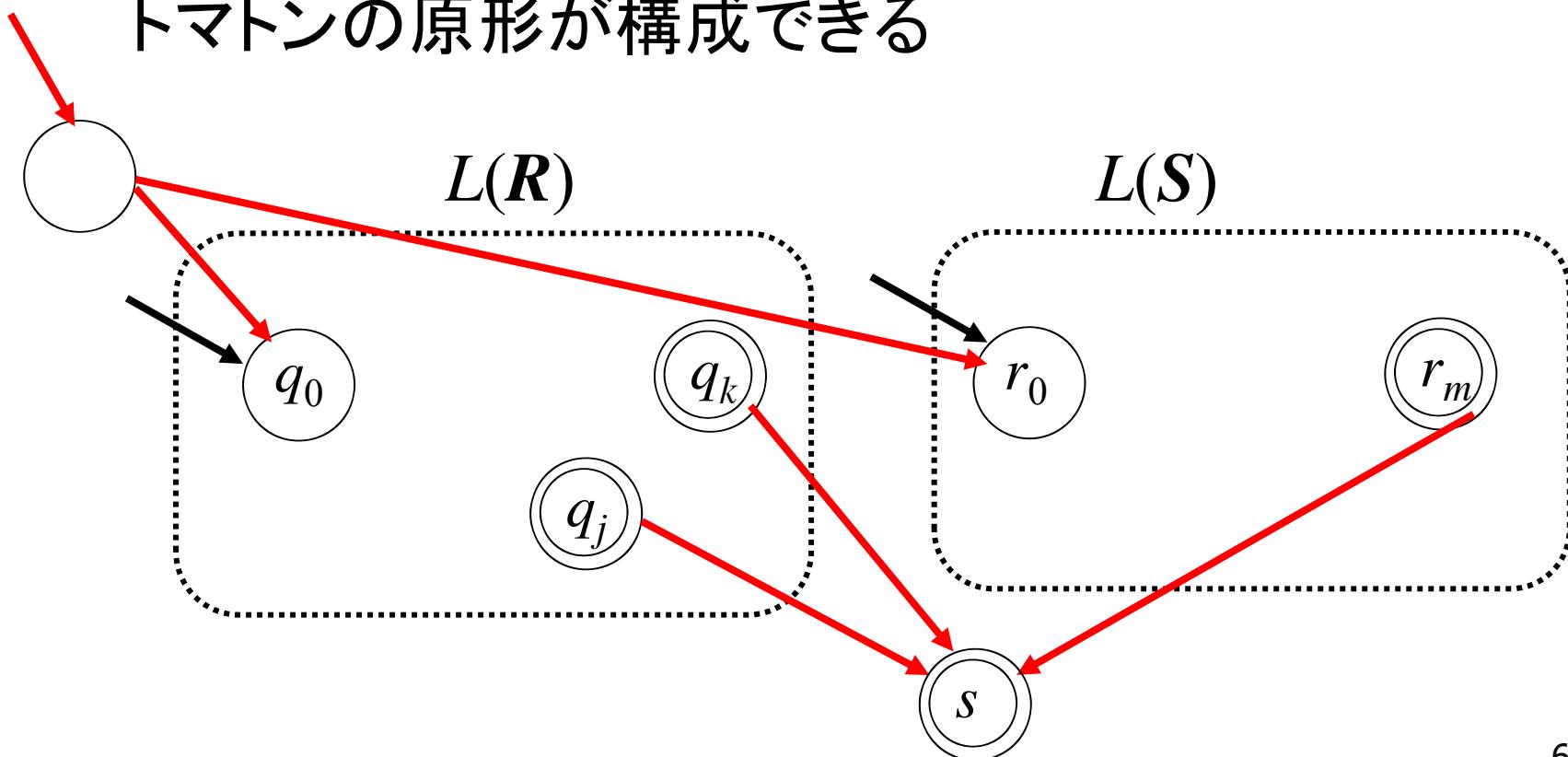
- 正則表現 $\emptyset, \varepsilon, c$ に対して $L(\emptyset), L(\varepsilon), L(c)$ を受理する有限状態オートマトンの原形は構成できる

$\Sigma = \{a, b\}$ のとき



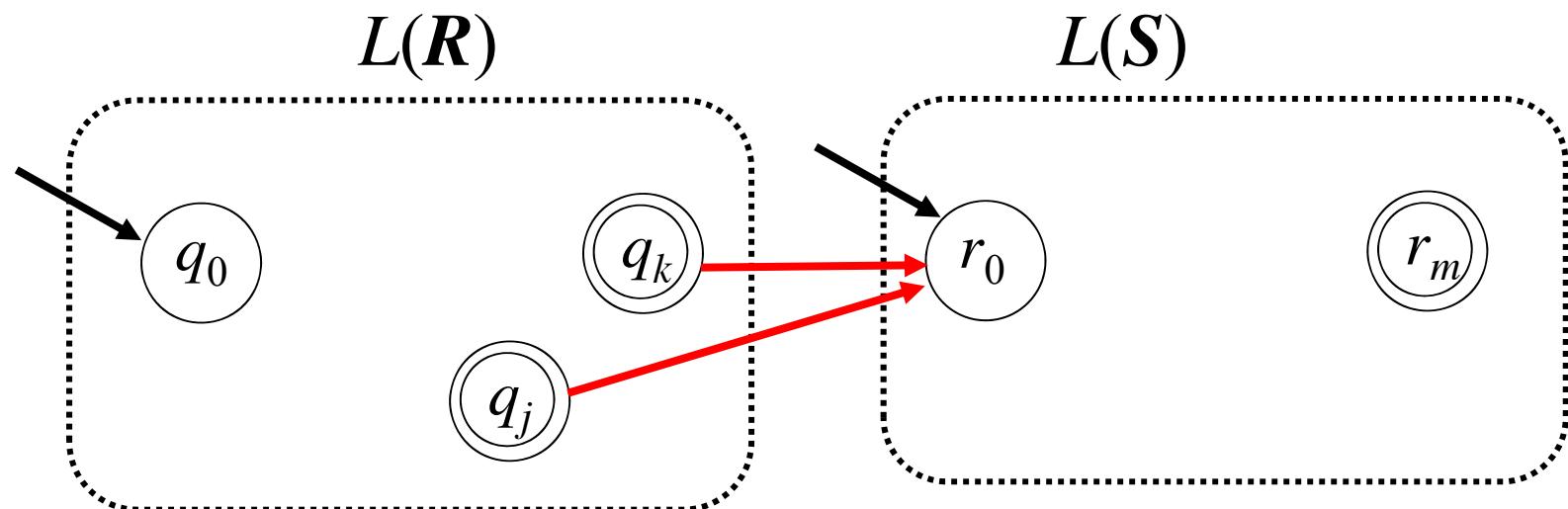
基本的なアイデア(2)

- 正則表現 R, S に対して $L(R), L(S)$ を受理する有限状態オートマトンの一部が構成できていれば、それらを“並列”に繋げば、 $L(R+S)$ を受理する有限状態オートマトンの原形が構成できる



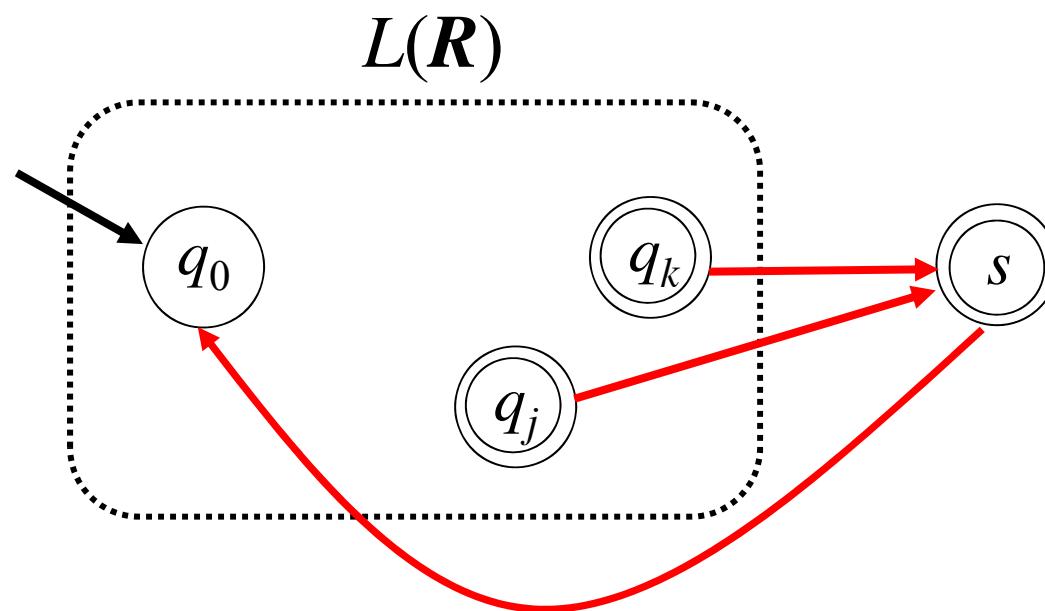
基本的なアイデア(3)

- 正則表現 R, S に対して $L(R), L(S)$ を受理する有限状態オートマトンの原形が構成できていれば、それらを“直列”に繋げば、 $L(RS)$ を受理する有限状態オートマトンの原形が構成できる



基本的なアイデア(3)

- 正則表現 R に対して $L(R)$ を受理する有限状態オートマトンの原形が構成できていれば、それを用いて“ループ”を構成すれば、 $L(R^*)$ を受理する有限状態オートマトンの原形が構成できる

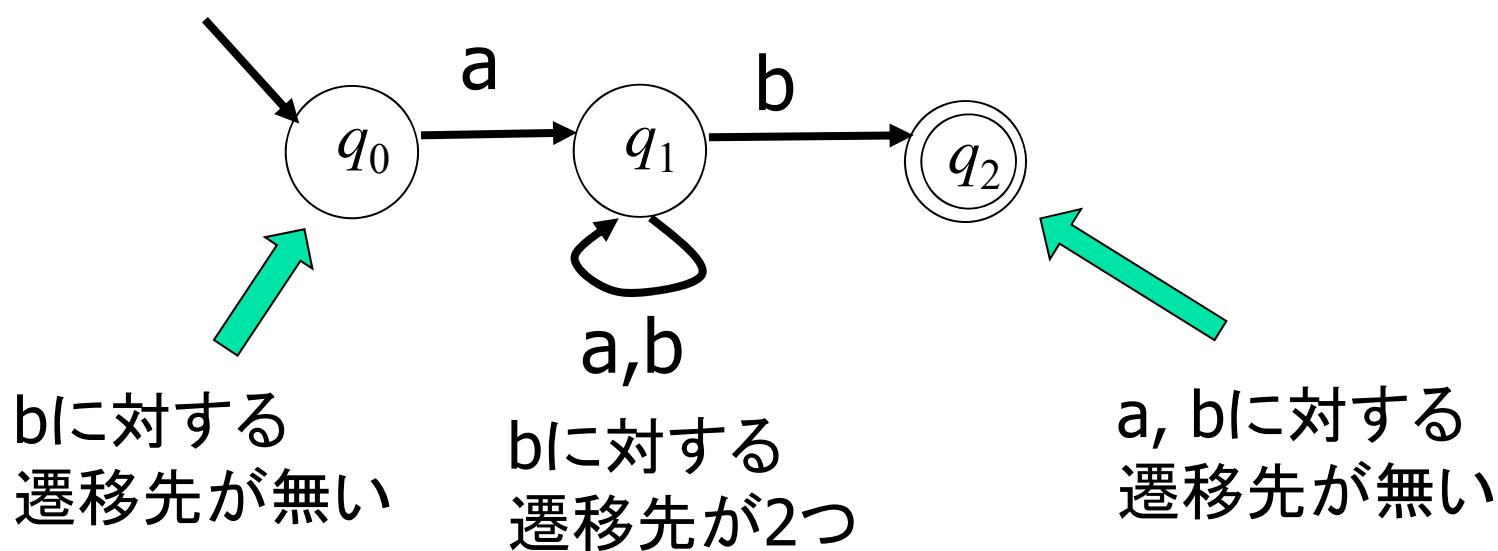


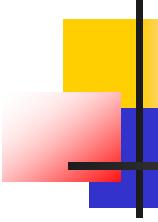
非決定性有限状態オートマトン

- 状態 q_i において記号 c が読み込まれたときの遷移先が
0個以上

Example

$$L(\mathbf{a}(\Sigma^*)\mathbf{b}) = \{ w \in \Sigma^* \mid w \text{は } a \text{ で始まり } b \text{ で終わる} \}$$





非決定性有限状態オートマトン

- 以下のような構成要素からなる組 $M=(\Sigma, S, \delta, q_0, F)$ を
非決定性有限状態オートマトン(NFA)とよぶ.
 - Σ はアルファベット
 - S は空でない集合であり,
その要素を状態とよぶ.
 - δ は $S \times \Sigma \rightarrow 2^S$ なる関数
 - δ は右のような表で表現される
 - $q_0 \in S$ は特定の状態であり, 初期状態とよぶ.
 - $F \subset S$ は特定の状態の集合であり, その要素を終了状態, もしくは受理状態とよぶ.

	F	a_1	\dots	a_n
q_0				
\dots				
q_m				

(

非決定性有限状態オートマトンが受理する記号列

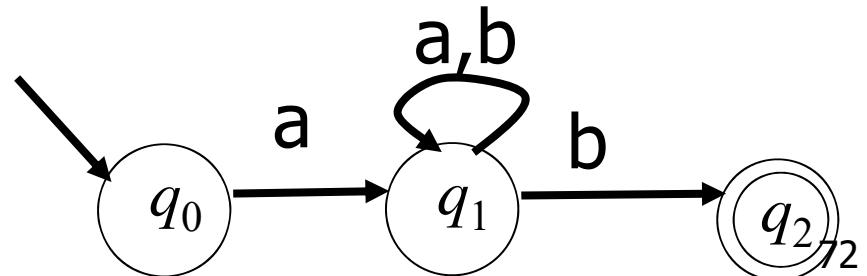
- 非決定性有限状態オートマトン $M = (\Sigma, S, \delta, q_0, F)$
- 記号列 $w = c_1c_2 \cdots c_k$ に対して
 $\delta(q_0, c_1) \ni q_1, \delta(q_1, c_2) \ni q_2, \dots, \delta(q_{k-1}, c_k) \ni q_k$ かつ $q_k \in F$
を満たす状態の列 q_1, q_2, \dots, q_k を構成できるとき、 M は w を受理するという

Example

abb : $\delta(q_0, a) \ni q_1, \delta(q_1, b) \ni q_1, \delta(q_1, b) \ni q_2, q_2 \in F$

aaa : $\delta(q_0, a) \ni q_1, \delta(q_1, a) \ni q_1, \delta(q_1, a) \ni q_1, q_2 \notin F$

b : $\delta(q_0, b) = \emptyset$

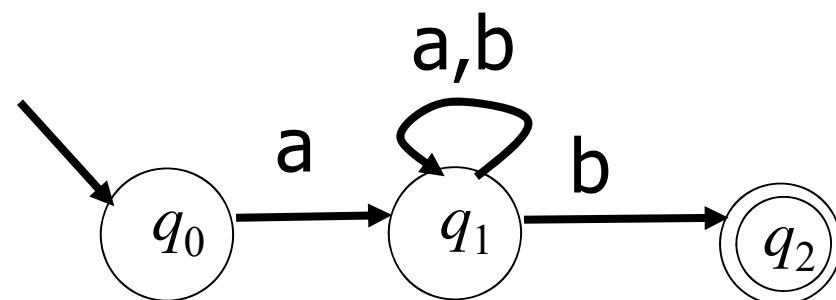


非決定性有限状態オートマトンが受理する言語

- 非決定性有限状態オートマトン $M = (\Sigma, S, \delta, q_0, F)$ に対して

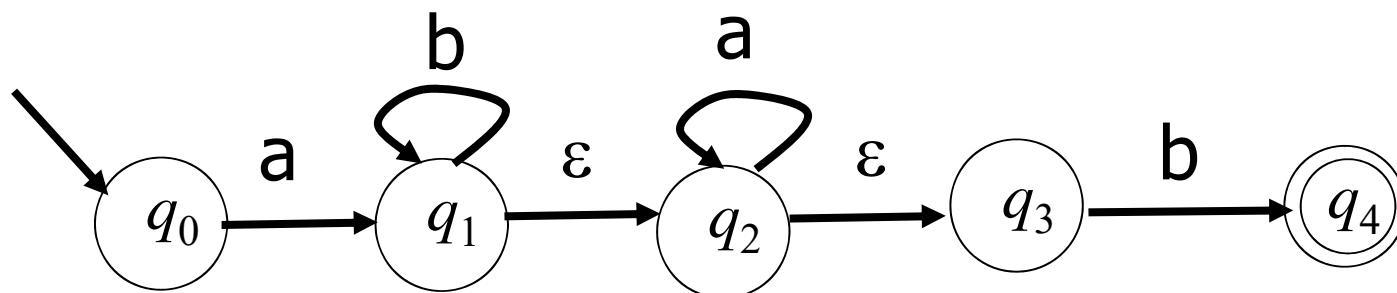
$L(M) = \{w \in \Sigma^* \mid w \text{ は } M \text{ によって受理される}\}$

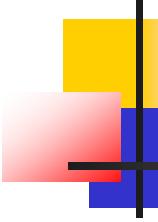
を M が受理する言語という。



ϵ 遷移

- 状態 q において記号が読み込まれなくても状態を遷移することを認めるとき、そのような遷移を ϵ 遷移という。
 - 記号を読む前後どちらでも好きなだけ遷移してよい





ϵ 遷移を持つ非決定性有限状態オートマトン

- 以下のような構成要素からなる組 $M = (\Sigma, S, \delta, q_0, F)$ を ϵ 遷移を持つ非決定性有限状態オートマトンとよぶ.
 - Σ はアルファベット
 - S は空でない集合であり,
その要素を状態とよぶ.
 - δ は $S \times (\Sigma \cup \{\epsilon\}) \rightarrow 2^S$ なる関数
 - δ は右のような表で表現される
 - $q_0 \in S$ は特定の状態であり, 初期状態とよぶ.
 - $F \subset S$ は特定の状態の集合であり, その要素を終了状態, もしくは受理状態とよぶ.

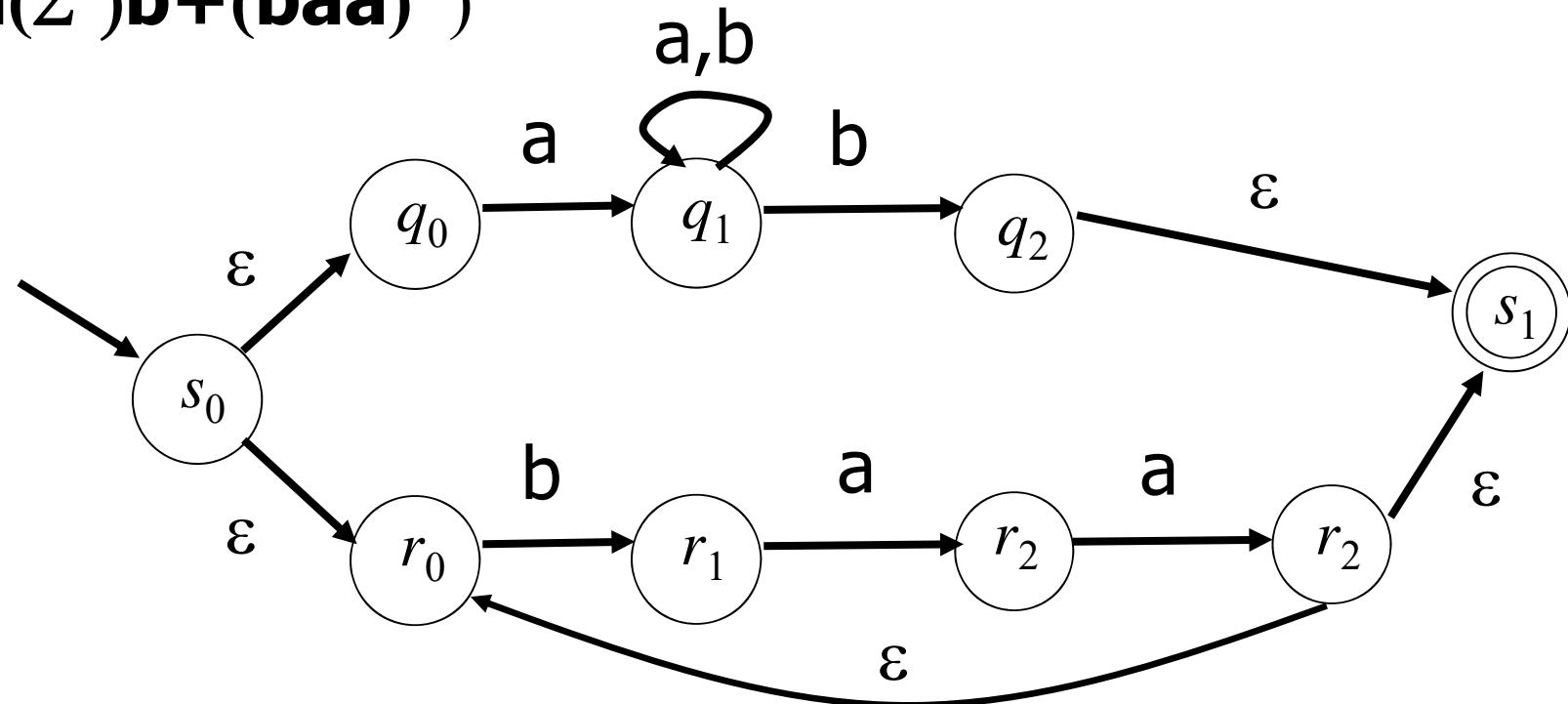
	F	a_1	\dots	a_n	ϵ
q_0					
\dots					
q_m					

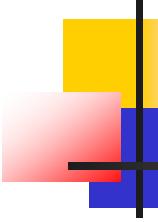
正則表現とオートマトン

- 正則表現を ϵ 遷移を持つ非決定性有限状態オートマトンへ変換する

Example

$$L(\mathbf{a}(\Sigma^*)\mathbf{b} + (\mathbf{baa})^*)$$





NFAからDFAへの変換の方針

- 部分集合構成法

ε 遷移を持つ非決定性オートマトン $M = (\Sigma, S, \delta, q_0, F)$

$$\delta: S \times (\Sigma \cup \{\varepsilon\}) \rightarrow 2^S$$



決定性オートマトン $M' = (\Sigma, 2^S, \delta', q_0, F)$

$$\delta': S \times \Sigma \rightarrow 2^S$$

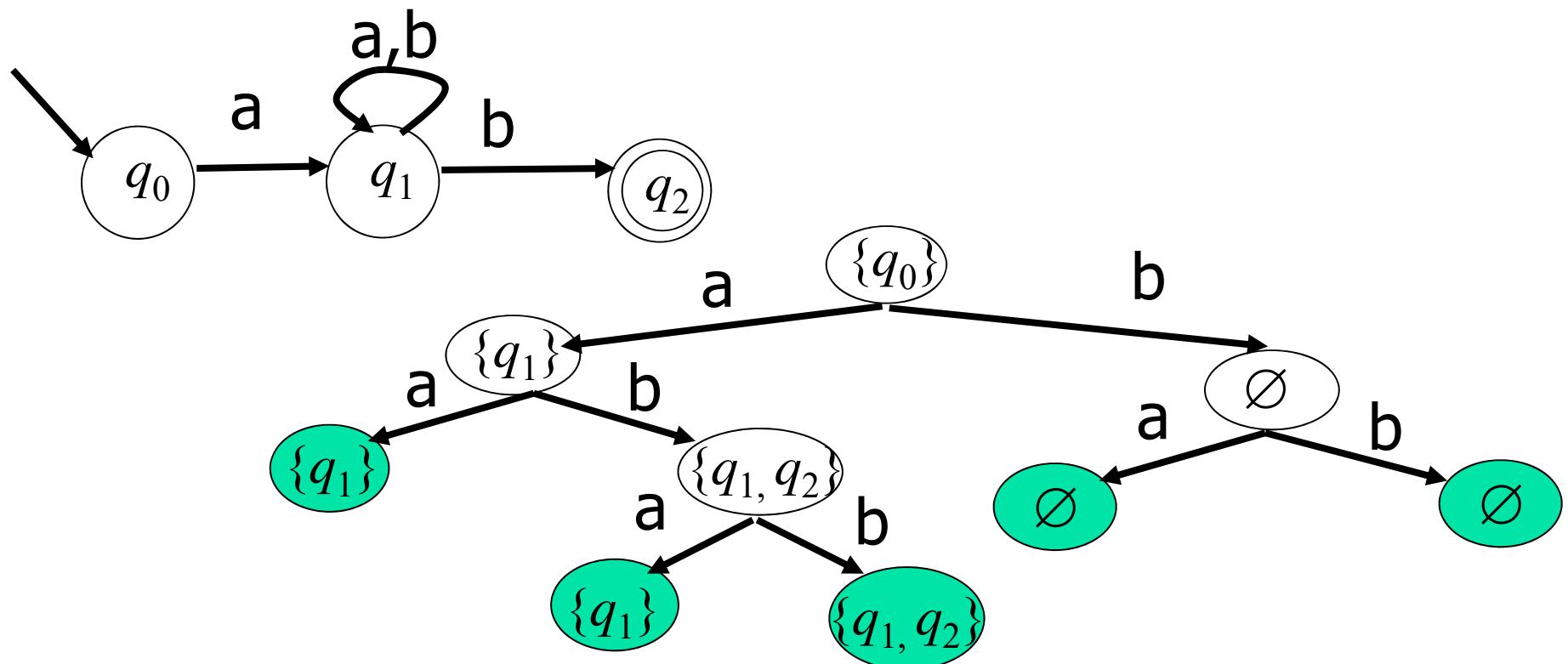
- 記法

非決定性オートマトン M の状態の集合 $Q = \{q_{i1}, \dots, q_{ik}\}$

$$\delta(Q, a) = \delta(q_{i1}, a) \cup \dots \cup \delta(q_{ik}, a)$$

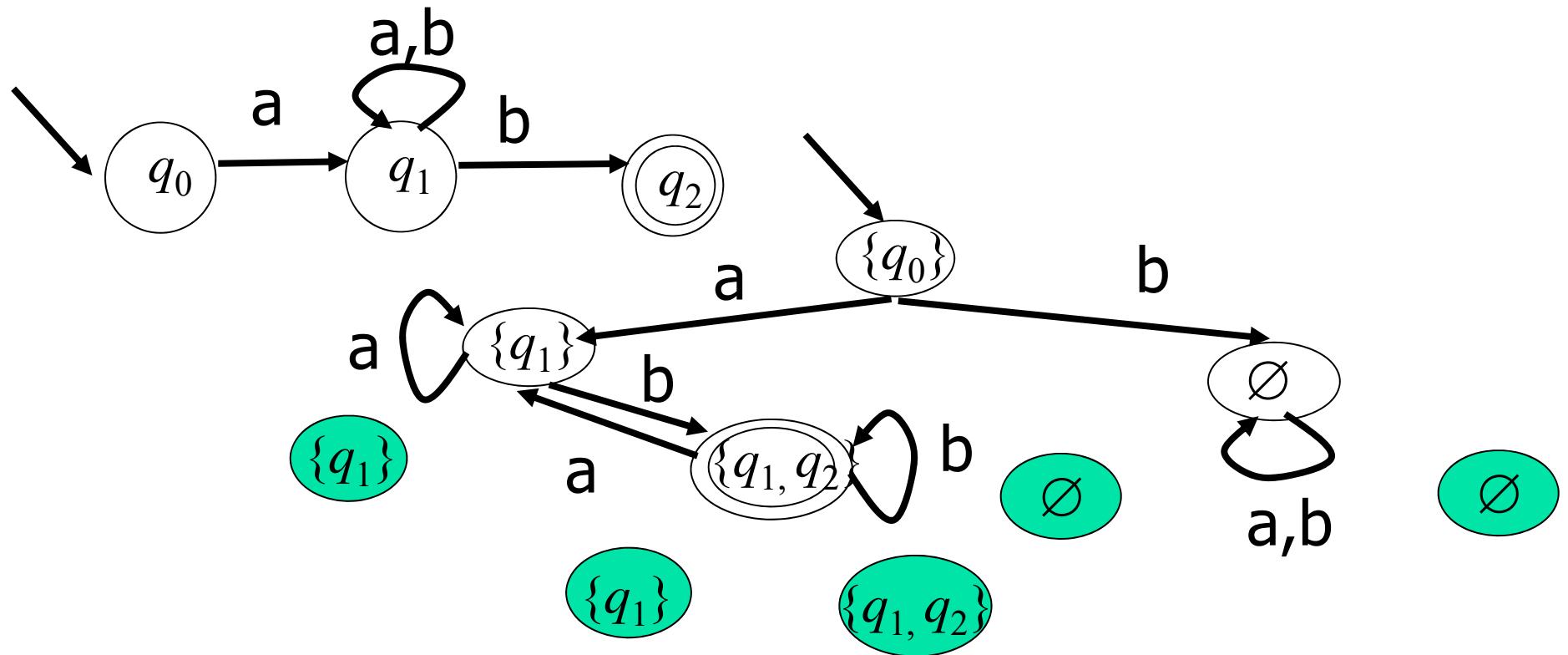
ϵ 遷移が無い場合

- 根に $\{q_0\}$ をラベル付けする. ラベルが Q である節点の子に $\delta(Q, a)$ をラベル付けする



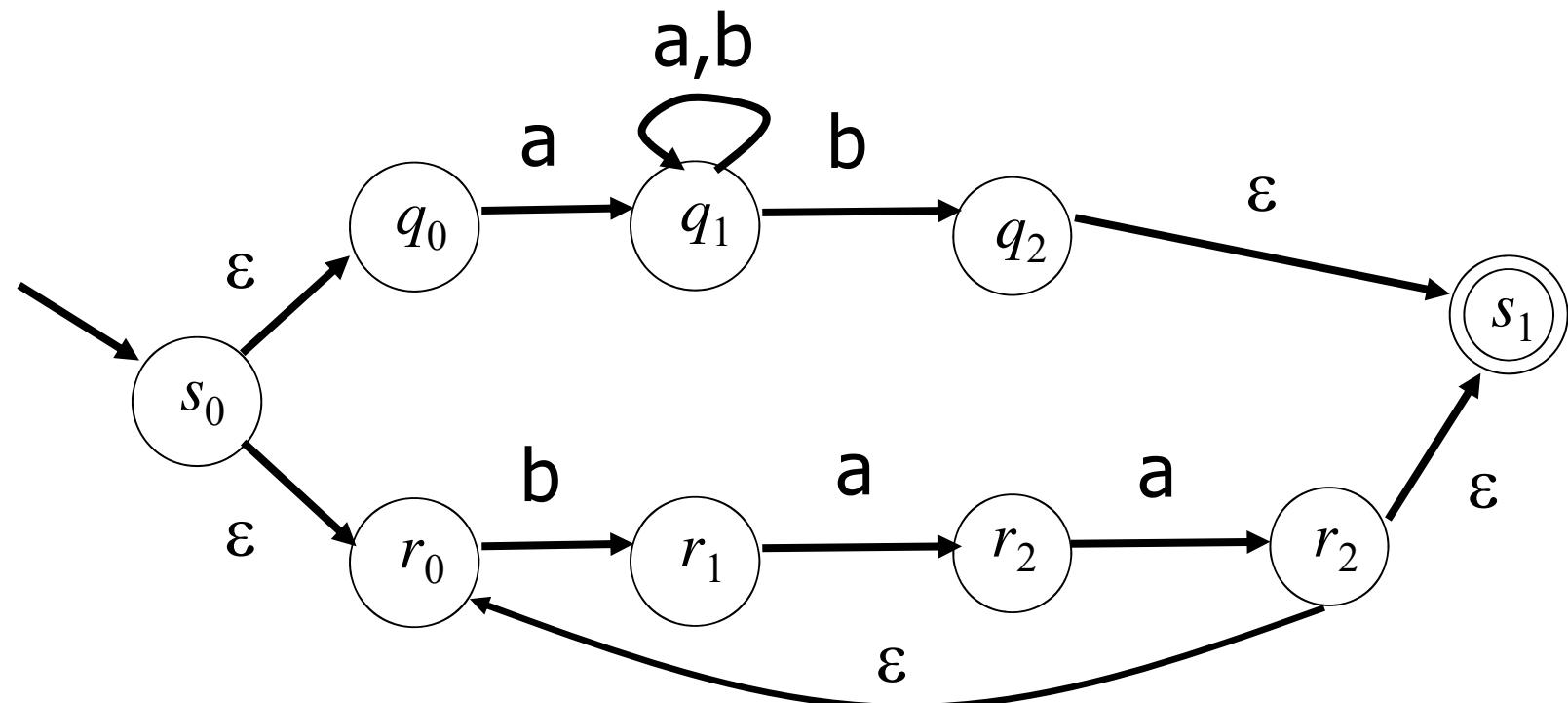
ϵ 遷移が無い場合

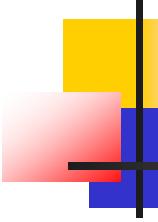
- 同じラベルのノードを一つにして状態遷移とする
- 受理状態を含むノードを受理状態にする



ε 閉包

- ε 遷移を持つ非決定性有限状態オートマトン $M = (\Sigma, S, \delta, s_0, F)$ において、状態 q から ε で遷移する先全体の集合 (q 自身を含む) を q の ε 閉包とよび、 $\varepsilon\text{-cl}(q)$ で表す。





ϵ 遷移の除去

- ϵ 遷移を持つ非決定性有限状態オートマトン $M = (\Sigma, S, \delta, s_0, F)$ に対して,

$$\delta'(q, c) = \epsilon\text{-cl}(\delta(\epsilon\text{-cl}(q)), c)$$

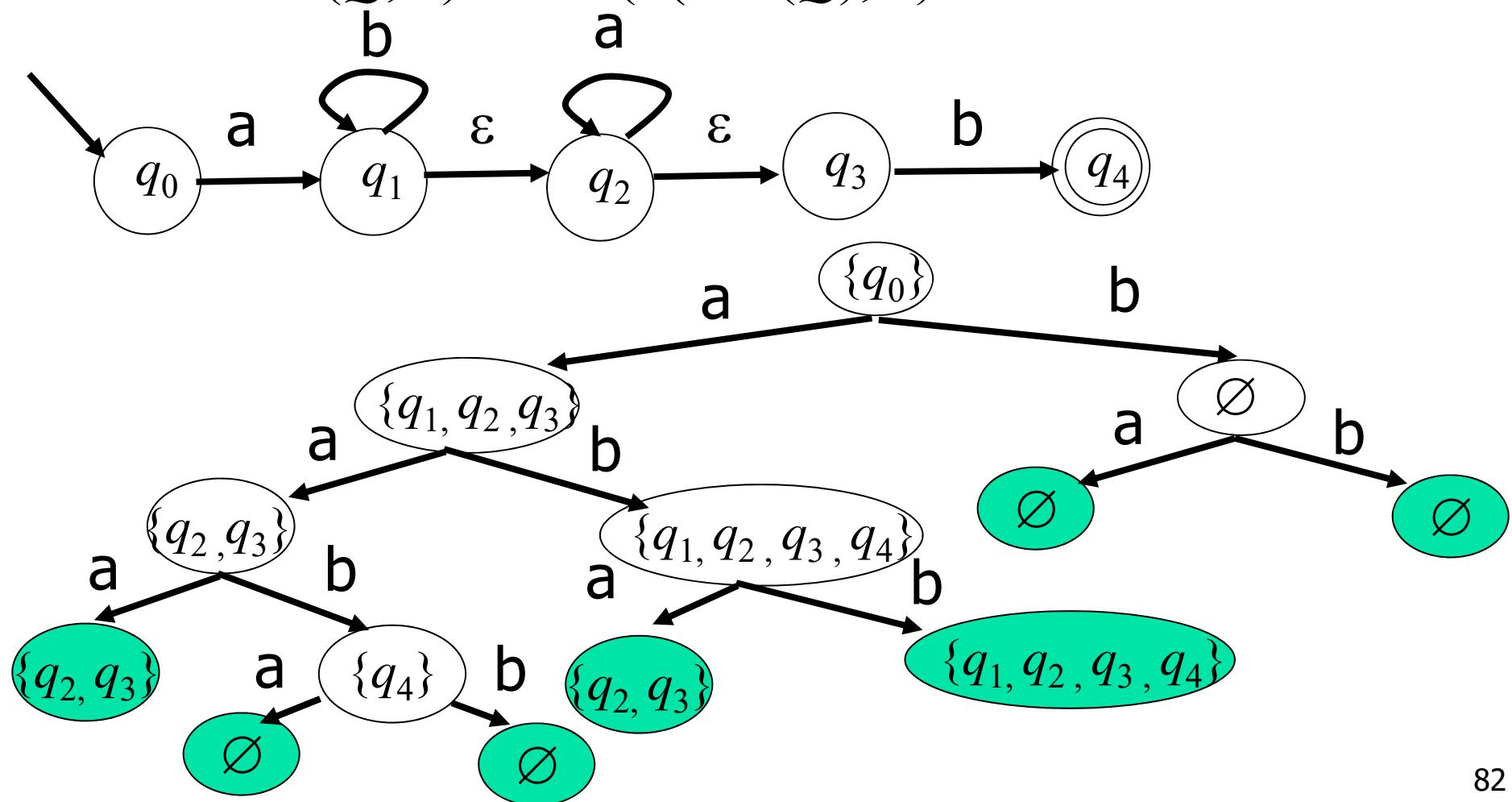
と定義して得られる非決定性有限状態オートマトンを $M' = (\Sigma, S, \delta', q_0, F)$ とすると $L(M) = L(M')$ が成立つ.

- 証明は記号列 w の長さに関する帰納法による

ε 遷移がある場合

訂正あり

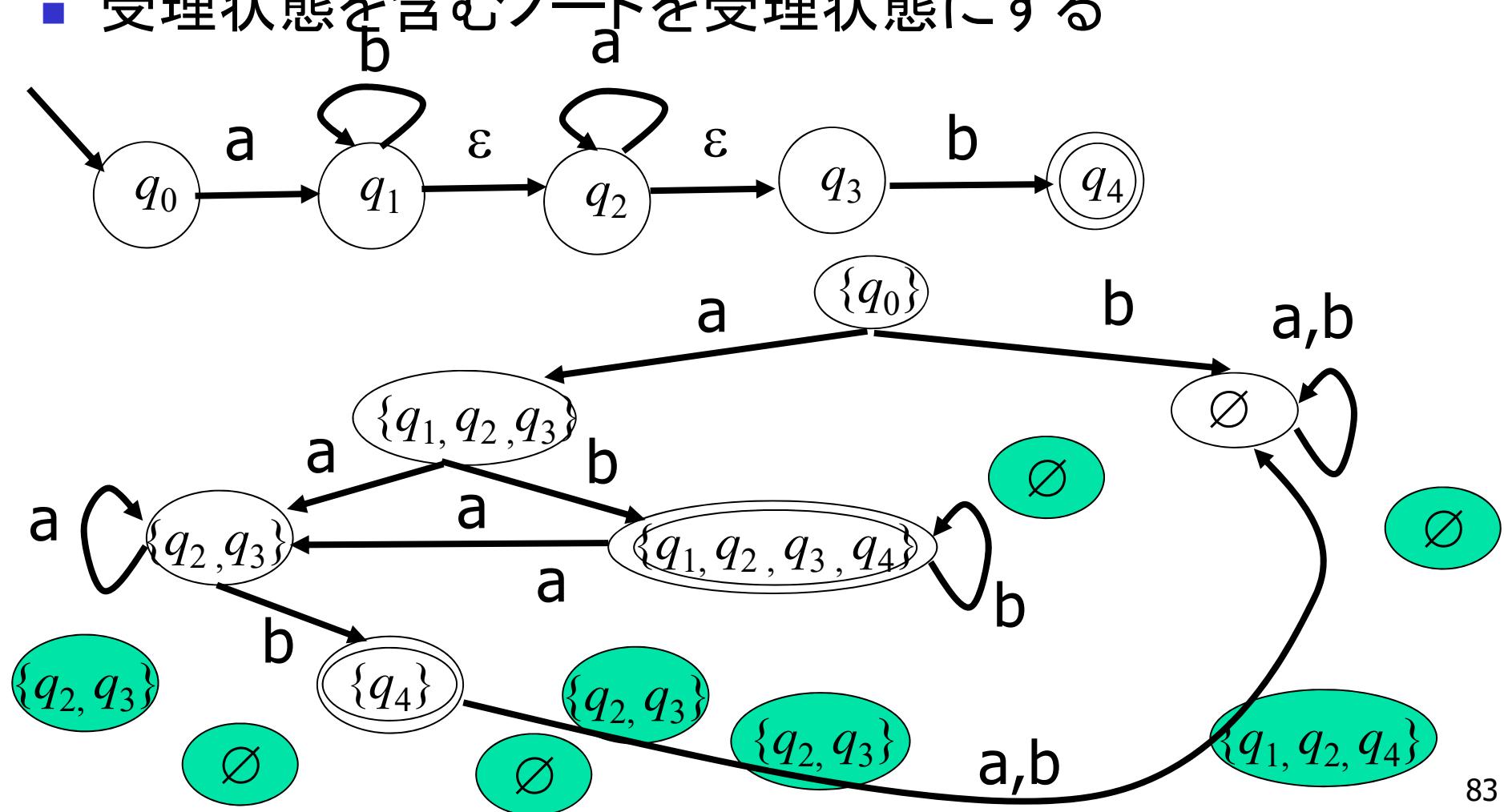
- 根に $\varepsilon\text{-cl}(\{q_0\})$ をラベル付けする. ラベルが Q である節点の子に $\delta'(Q, c) = \varepsilon\text{-cl}(\delta(\varepsilon\text{-cl}(Q), c)$ をラベル付けする



ϵ 遷移がある場合

訂正あり

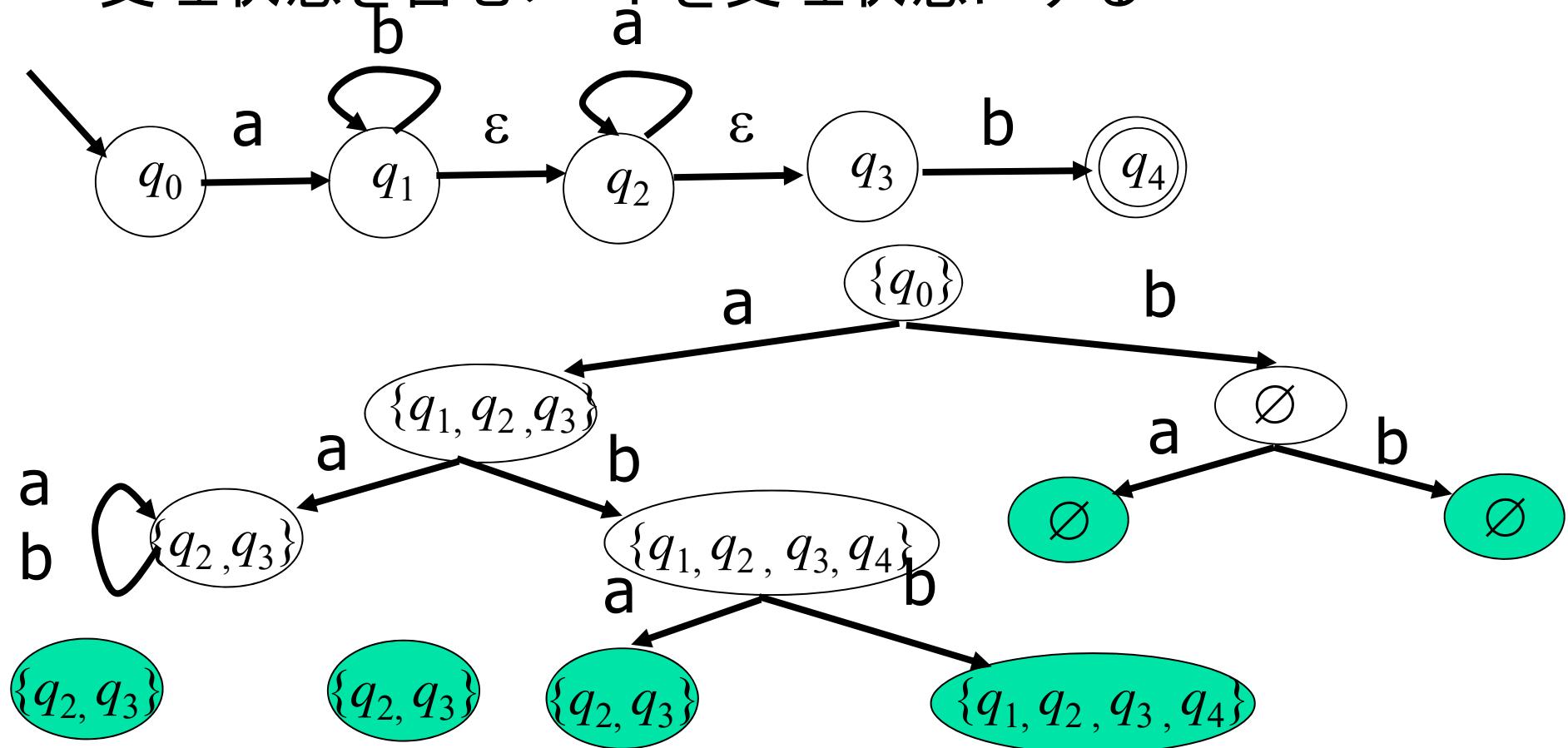
- 同じラベルのノードを一つにして状態遷移とする
- 受理状態を含むノードを受理状態にする



ϵ 遷移がある場合

不要
(28ページと同じ)

- 同じラベルのノードを一つにして状態遷移とする
- 受理状態を含むノードを受理状態にする





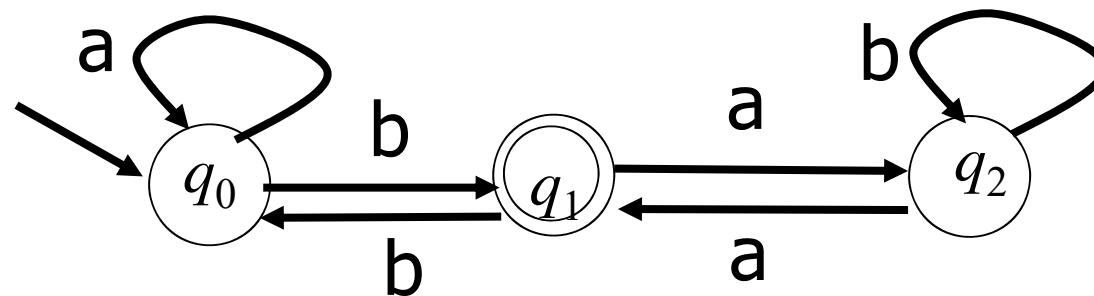
有限状態オートマトン

正則表現

~~正則表現から有限状態オートマトン
への変換~~

McNorton-Yamadaの方法(1)

- Step -1. q_i から q_j へ直接遷移する記号だけからなる集合を正則表現で表す. ただし, $i=j$ については ε も加える



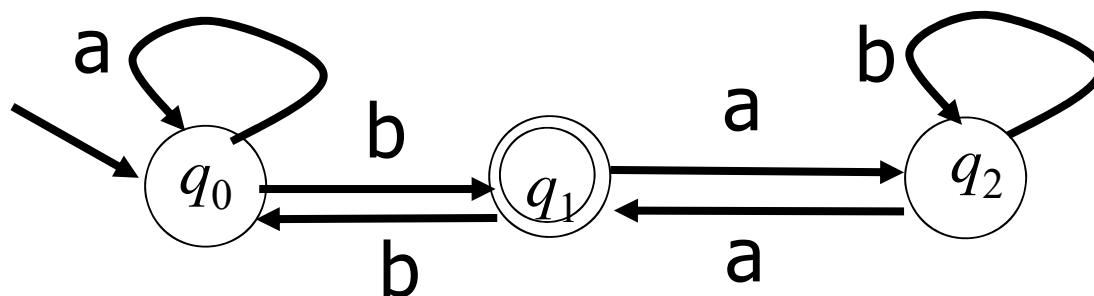
$$R_{00} = \mathbf{a} + \varepsilon, R_{11} = \emptyset + \varepsilon = \varepsilon, R_{22} = \mathbf{b} + \varepsilon,$$

$$R_{01} = \mathbf{b}, R_{02} = \emptyset, R_{10} = \mathbf{b}, R_{12} = \mathbf{a},$$

$$R_{20} = \emptyset, R_{21} = \mathbf{a},$$

McNorton-Yamadaの方法(2)

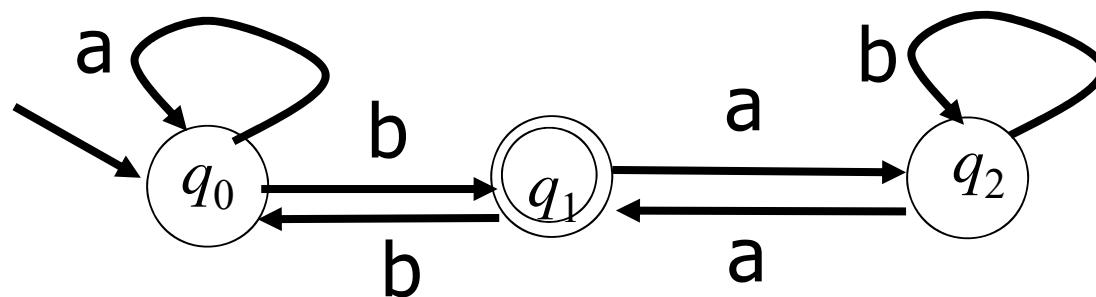
- Step 0. q_i から q_j へ直接遷移するか, または, q_0 だけを通過して遷移する記号列からなる集合を表す正則表現 R_{ij}^0 を求める



$$\begin{aligned} R_{11}^0 &= R_{11} + R_{10}(R_{00})^* R_{01} \\ &= (\varepsilon) + (\mathbf{b} + \varepsilon) (\mathbf{a} + \varepsilon)^* (\mathbf{b} + \varepsilon) \\ &= \varepsilon + \mathbf{b} \mathbf{a}^* \mathbf{b} \end{aligned}$$

McNorton-Yamadaの方法(3)

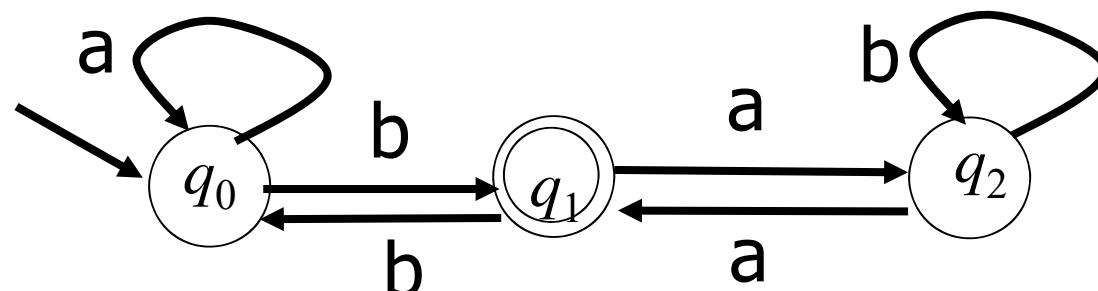
- Step 0. q_i から q_j へ直接遷移するか, または, q_0 だけを通過して遷移する記号列からなる集合を表す正則表現 R_{ij}^0 を求める



$$\begin{aligned} R_{12}^0 &= R_{12} + R_{10}(R_{00})^* R_{02} \\ &= \mathbf{a} + (\mathbf{b} + \varepsilon) (\mathbf{a} + \varepsilon)^* \emptyset \\ &= \mathbf{a} \end{aligned}$$

McNorton-Yamadaの方法(4)

- Step 1. q_i から q_j へ直接遷移するか, または, q_0 と q_1 を通過して遷移する記号列からなる集合を表す正則表現 R_{ij}^k を求める

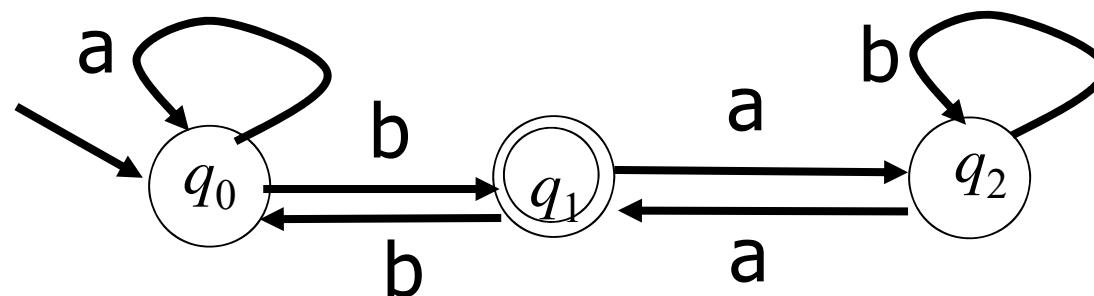


q0 or q1

$$\begin{aligned} R_{02}^1 &= R_{02}^0 + R_{01}^0 (R_{11}^0)^* R_{12}^0 = \\ &= \emptyset + (\mathbf{a}^* \mathbf{b}) (\varepsilon + \mathbf{b} \mathbf{a}^* \mathbf{b})^* (\mathbf{a}) \\ &= \mathbf{b} + \mathbf{a}^* \mathbf{b} \mathbf{a} + \varepsilon + \mathbf{a}^* \mathbf{b} (\mathbf{b} \mathbf{a}^* \mathbf{b})^* \mathbf{a} \end{aligned}$$

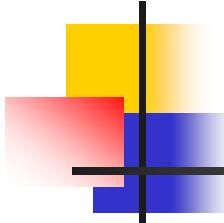
McNorton-Yamadaの方法(5)

- Step 2. q_i から q_j へ直接遷移するか, または, q_0, q_1, q_2 を通過して遷移する記号列からなる集合を表す正則表現 R^2_{ij} を求める



$$R^2_{02} = R^1_{02} + R^1_{01} \underbrace{(R^1_{11})^*}_{\substack{02 \\ 22}} R^1_{12}$$

q0 or q1 or q2



McNorton-Yamadaの方法

有限状態オートマトンの状態数を N とする

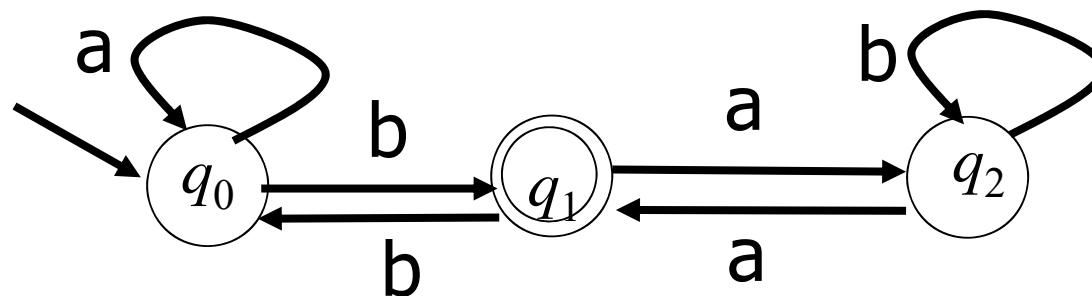
- Step -1. q_i から q_j へ直接遷移する記号だけからなる正則表現 R_{ij} を求める. ただし, $i=j$ については ε も加える

$k = 0, \dots, N-1$ について順に

- Step k . q_i から q_j へ直接遷移するか, または, q_0, \dots, q_k を通過して遷移する記号列の集合を表す正則表現 R^k_{ij} を求める

有限オートマトンから正則表現へ

- Step 0. q_i から q_j へ直接遷移するか, または, q_0 だけを通過して遷移する記号列を正則表現にする



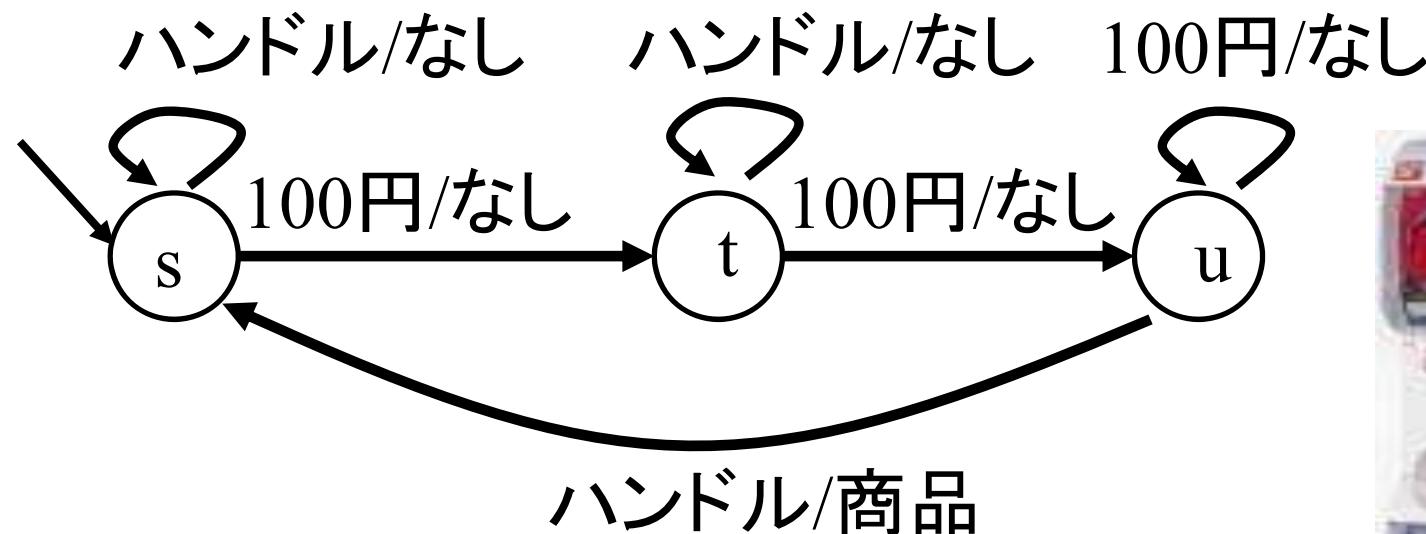
$$R^0_{01} = R_{01} + R_{00}(R_{00})^* R_{10} =$$

$$R^0_{11} = R_{11} + R_{10}(R_{00})^* R_{01} =$$

$$\begin{aligned} R^0_{00} &= R_{00} + R_{00}(R_{00})^* R_{00} \\ &= (\mathbf{a} + \varepsilon) + (\mathbf{a} + \varepsilon)(\mathbf{a} + \varepsilon)^* (\mathbf{a} + \varepsilon) \end{aligned}$$

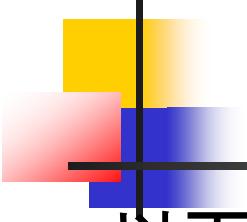
Mealy型順序機械

- 100円硬貨専用で200円商品の自動販売機



	100円	ハンドル
s	t/なし	s/なし
t	u/なし	t/なし
u	u/なし	s/商品





Mealy型順序機械の表現

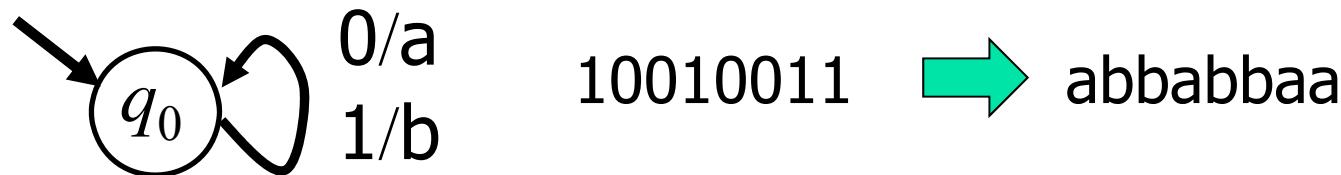
- 以下のような構成要素からなる組 $M = (\Sigma, \Delta, S, \delta, \lambda, q_0)$ をMealy型順序機械とよぶ。
 - Σ は入力アルファベット
 - Δ は出力アルファベット
 - S は空でない集合であり、その要素を状態とよぶ。
 - δ は $S \times \Sigma \rightarrow S$ なる関数
 - λ は $S \times \Sigma \rightarrow \Delta$ なる関数
 - δ と λ は、組 $(\delta(q_i, a_j), \lambda(q_i, a_j)) = (q_{ij}, c_{ij})$ を用いて上のような表で表現される
 - $q_0 \in S$ は特定の状態であり、初期状態とよぶ
 - 順序回路などでは初期状態を指定しない

	a_1	\dots	a_n
q_0	(q_{01}, c_{01})		(q_{0n}, c_{0n})
\dots			
q_m	(q_{m1}, c_{m1})		(q_{mn}, c_{mn})

Mealy型順序機械の例(1)

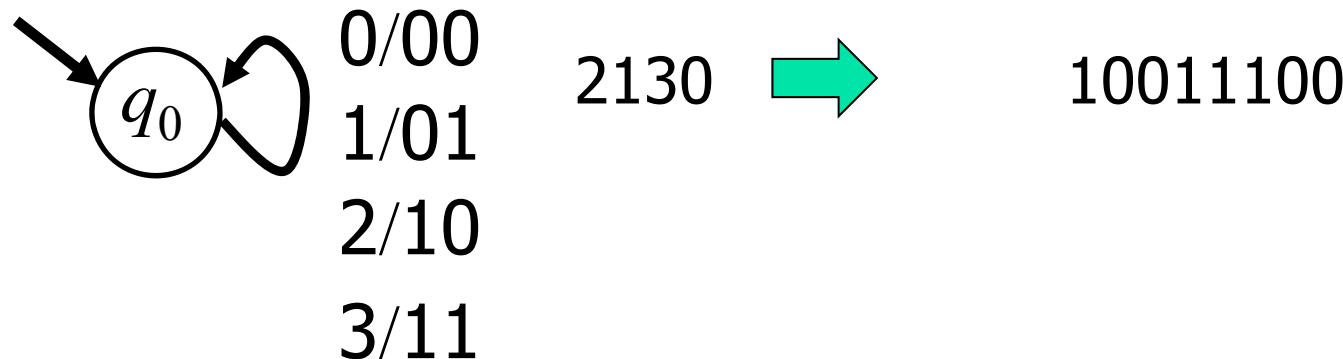
- 記号の変換

$$\Sigma = \{0,1\}, \Delta = \{a, b\}$$



10010011 → abbabbaa

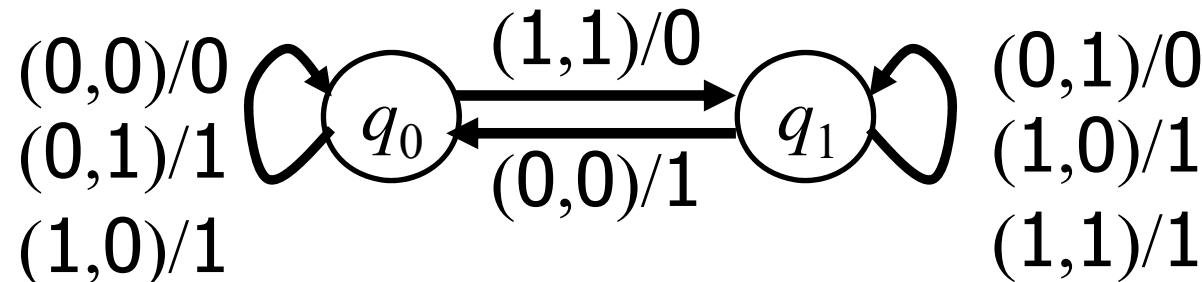
- 記号の変換 $\Sigma = \{0,1,2,3\}, \Delta = \{00, 01, 10, 11\}$



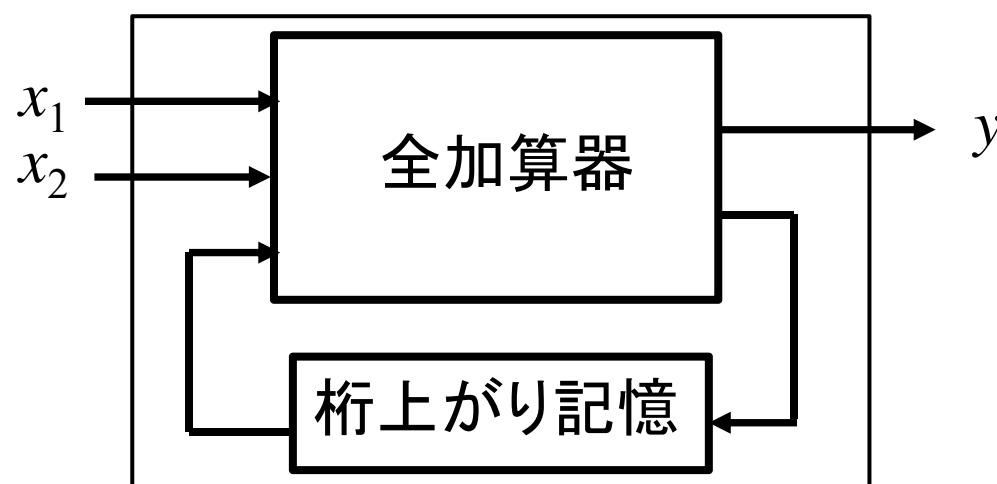
2130 → 10011100

Mealy型順序機械の例(2)

- 直列加算器 $\Sigma = \{(0,0), (0,1), (1,0), (1,1)\}$, $\Delta = \{0,1\}$

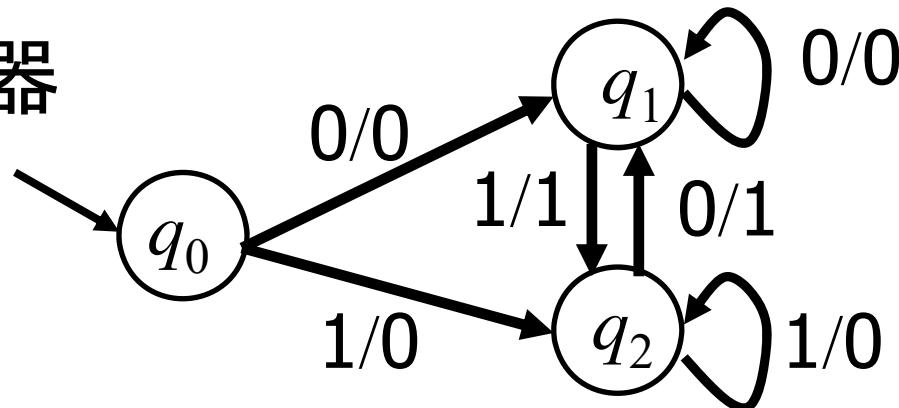


- 入力 x_1, x_2 には2つの自然数の2進表記が下位の桁から順に送られる



Mealy型順序機械の例(3)

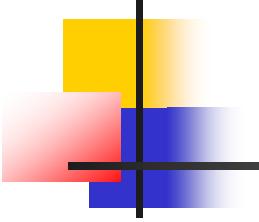
エッジ検出器



0000000000	→	0000000000
0001111100	→	0001000010
0001111100	→	0001000010
0000111000	→	0000100100
0000111000	→	0000100100
0000010000	→	0000011000
0000010000	→	0000011000
0000000000	→	0000000000

Mealy型順序機械の例(4)





Moore型順序機械

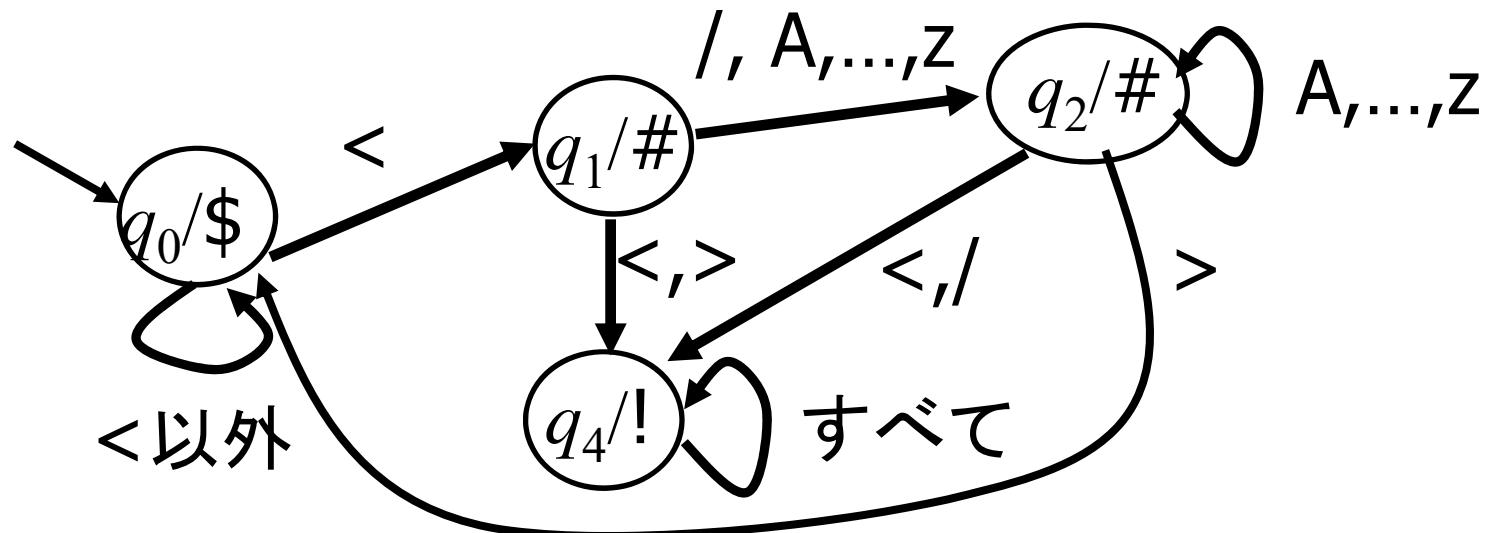
- 以下のような構成要素からなる組 $M=(\Sigma, \Delta, S, \delta, \lambda, q_0)$ をMoore型順序機械とよぶ。
 - Σ は入力アルファベット
 - Δ は出力アルファベット
 - S は空でない集合であり、その要素を状態とよぶ。
 - δ は $S \times \Sigma \rightarrow S$ なる関数
 - λ は $S \rightarrow \Delta$ なる関数
 - δ と λ は右のような表で表現される
 - $q_0 \in S$ は特定の状態であり、初期状態とよぶ
 - 順序回路などでは初期状態を指定しない
 - $\Delta=\{0, 1\}$ のとき 言語を受理する有限状態オートマトンとみなすことができる

	a_1	\dots	a_n	λ
q_0	q_{01}		q_{0n}	c_0
\dots				
q_m	q_{m1}		q_{mn}	c_m

Moore型順序機械の例(1)

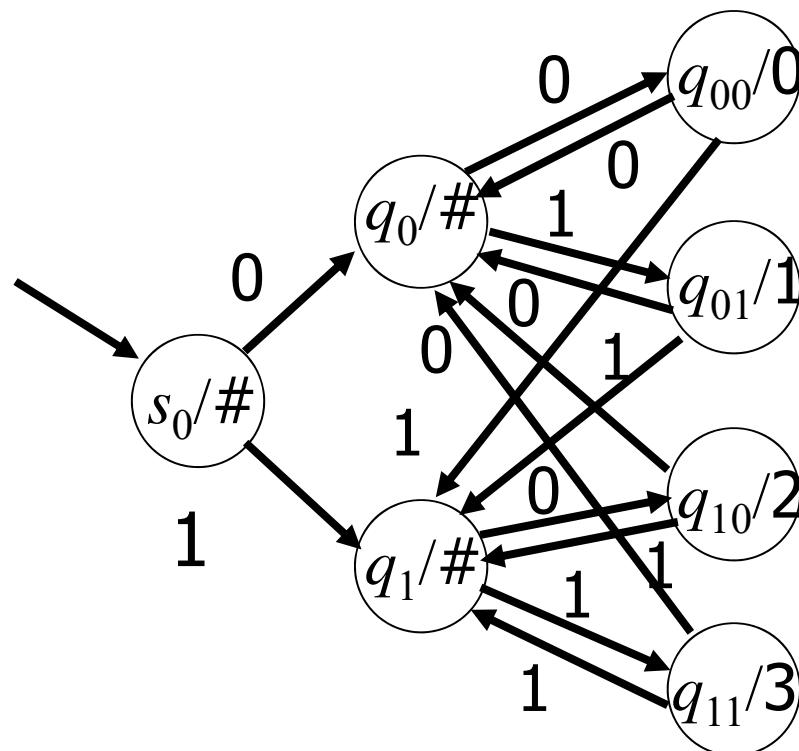
- 字句(トークン)解析 $\Sigma = \{<, >, /, A, \dots, Z\}$,
 $\Delta = \{\#, \$, !\}$
 - プログラミング言語などでは、処理の最小単位を**トークン**とよぶ

例 HTMLのタグは‘<’と‘>’で挟まれた記号列
または‘</’と‘>’で挟まれた記号列または‘</>’



Moore型順序機械の例(2)

- 2進表現から4進表現への変換



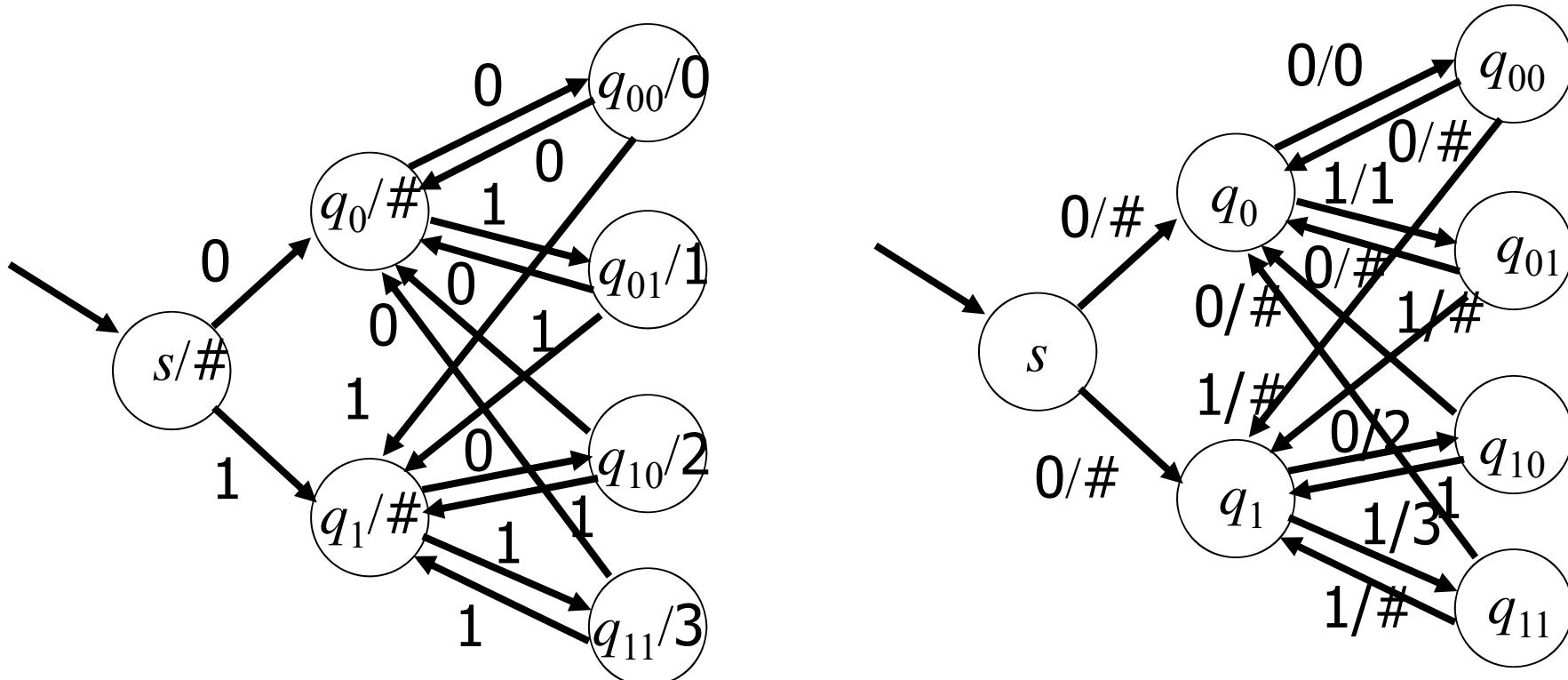
10010011



##2#1#0#3

例

- 2進表現から4進表現への変換



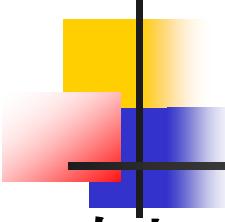
10010011



#2#1#0#3

Moore型順序機械の例(3)





Moore型からMealy型への変換

もとのMoore型を $M = (\Sigma, \Delta, S, \delta, \lambda, q_0)$ とし

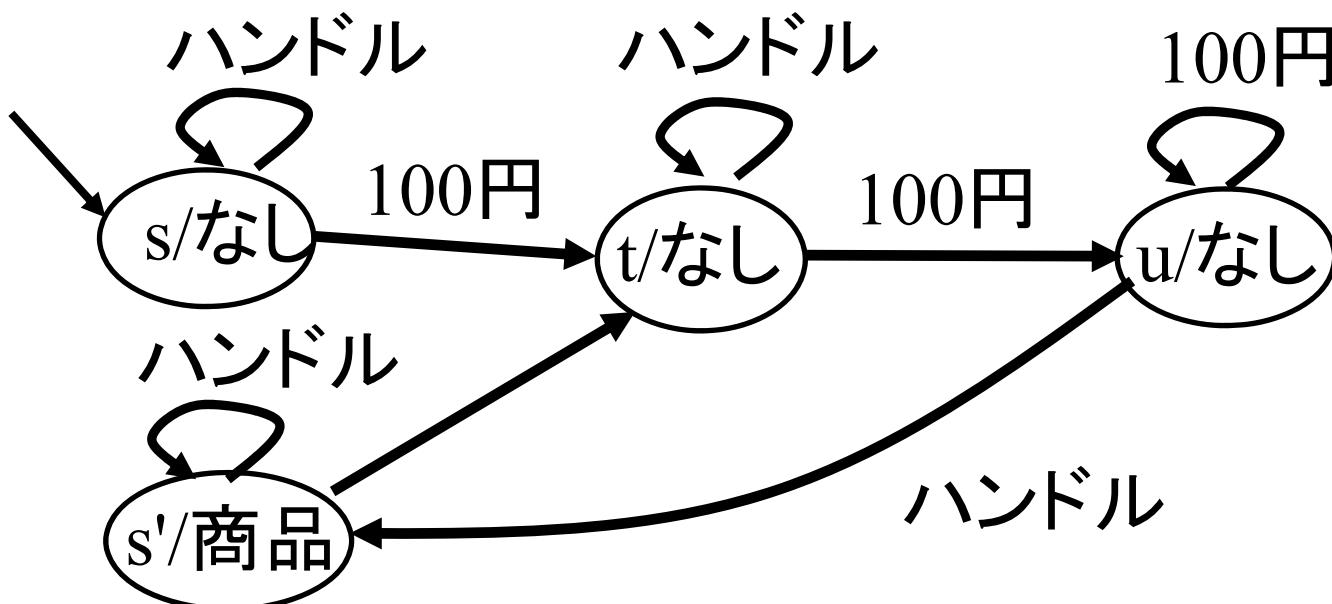
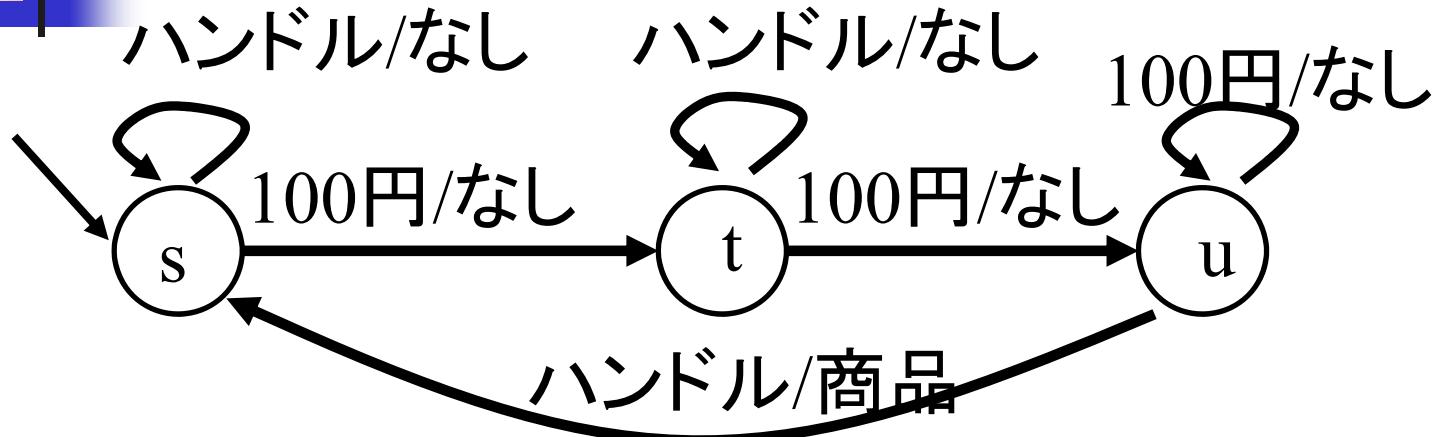
- 出力のタイミングを前にずらす
 - 記号列を読み込む直前の初期状態の出力を取り除く
- を認めれば、

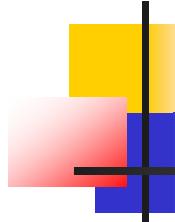
状態 q での出力 $\lambda(q)$ を、 q に至る遷移において出力する

$$\lambda'(q, a) = \lambda(\delta(q, a))$$

とすれば、Mealy型 $M' = (\Sigma, \Delta, S, \delta, \lambda', q_0)$ に変換できる

Mealy型順序機械とMoore型順序機械





Mealy型からMoore型への変換

もとのMealy型を $M = (\Sigma, \Delta, S, \delta, \lambda, q_0)$ とし, 求めるMoore型を $M' = (\Sigma, \Delta, S', \delta', \lambda', q_0)$ とする

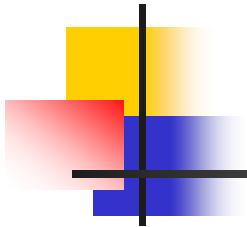
- 出力のタイミングを後ろにずらす
- 記号列を読み込む直前の初期状態の出力を定義しないを認める.
- M の状態と出力記号の組を新たに状態と考える

$$S' = S \times \Delta$$

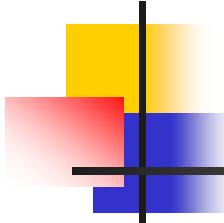
M が状態 q に至る遷移において c を出力するとすれば,
 M' においては (q, c) に遷移して c を出力する, とする

$$\delta'((q, c), a) = (\delta(q, a), \lambda(\delta(q, a)))$$

$$\lambda'((q, c)) = c$$



オートマトンによるキーワード探索



キーワード探索とは?

- テキスト探索, パターン検出, パターン照合, 記号列照合などともよばれる

入力: (長い)記号列 s , (短い)キーワード w

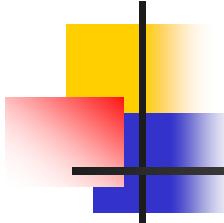
出力: 記号列 s で w が出現したすべての位置

例 $s = \text{asukaratokyotokyotoniiiku}$

$w = \text{tokyoto}$

asukaratokyotokyotoniiiku

00000000000010000100000



前綴り(prefix)と後綴り(suffix)

キーワード $w = c_1c_2c_3\dots c_k$

- (キーワードの)前綴り $c_1c_2c_3\dots c_j \ j=1,\dots,k$
tokyoto の前綴り t, to, tok, toky,..., tokyoto
- (キーワードの)後綴り $c_jc_{j+1}c_{j+2}\dots c_k \ j=1,\dots,k$
tokyoto の前綴り tokyoto, okyoto,...,oto,to,o

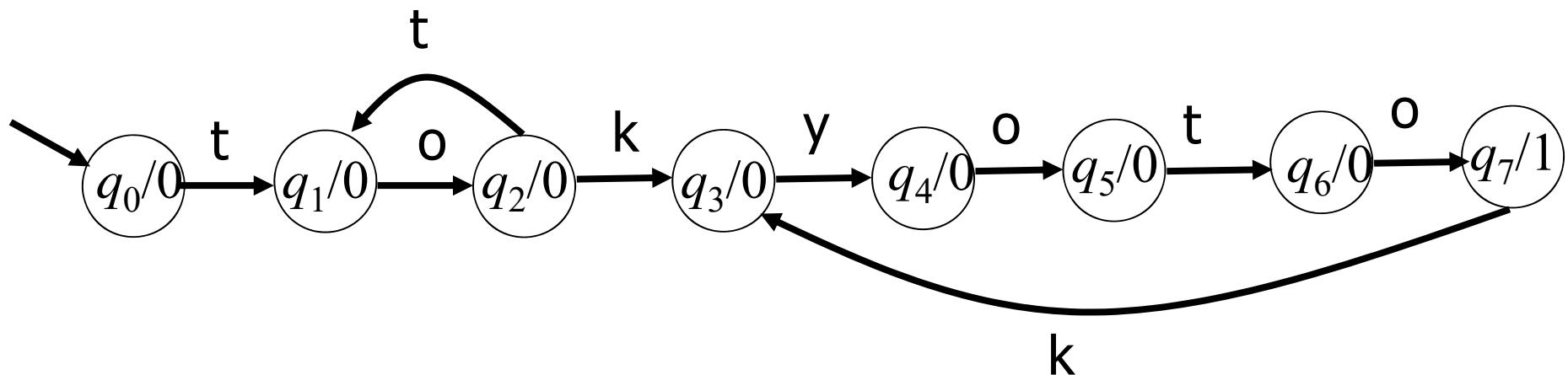
前綴り(prefix)から順序機械へ

キーワード $w = \text{tokyoto}$

入力記号 c を加えたときの前綴りの前綴りと後綴りに注目

tokt

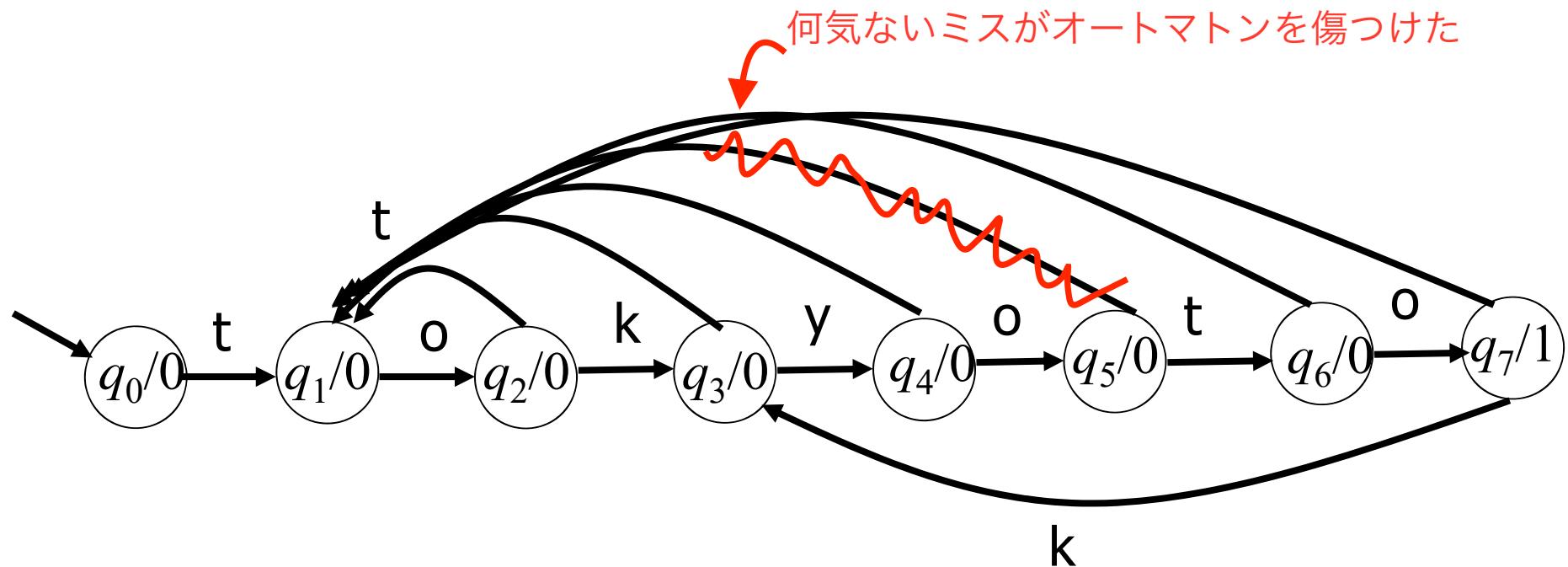
tokyotok



前綴り(prefix)から順序機械へ

キーワード $w = \text{tokyoto}$

入力記号 c を加えたときの前綴りの前綴りと後綴りに注目

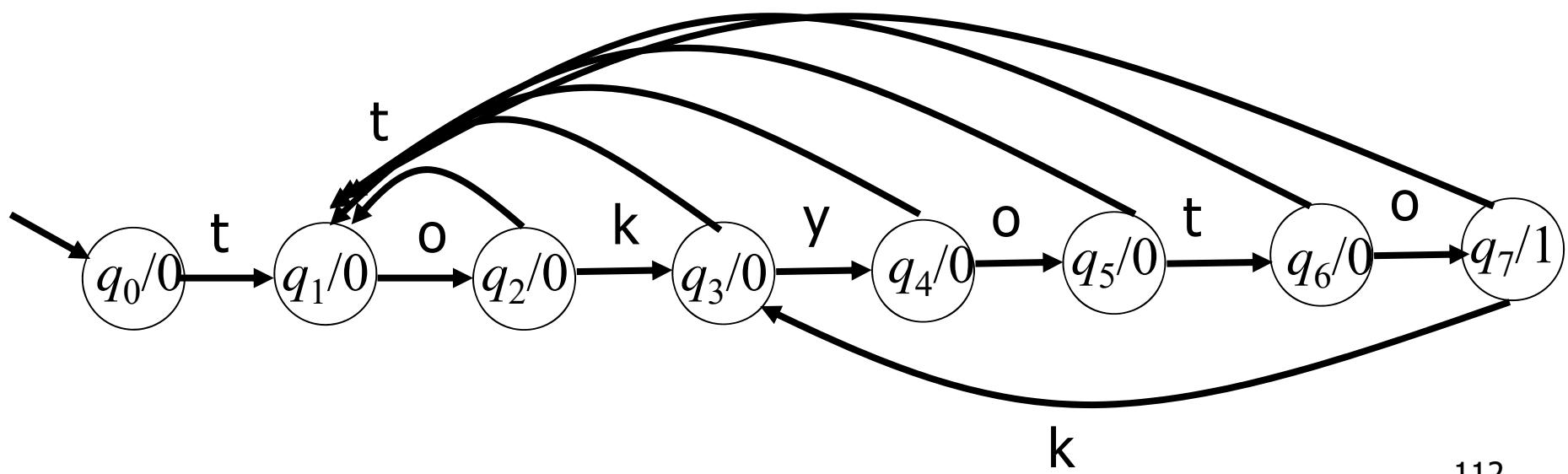


前綴り(prefix)から順序機械へ

キーワード $w = \text{tokyoto}$

入力記号 c を加えたときの前綴りの前綴りと後綴りに注目

図示した遷移以外はすべて q_0 に遷移する

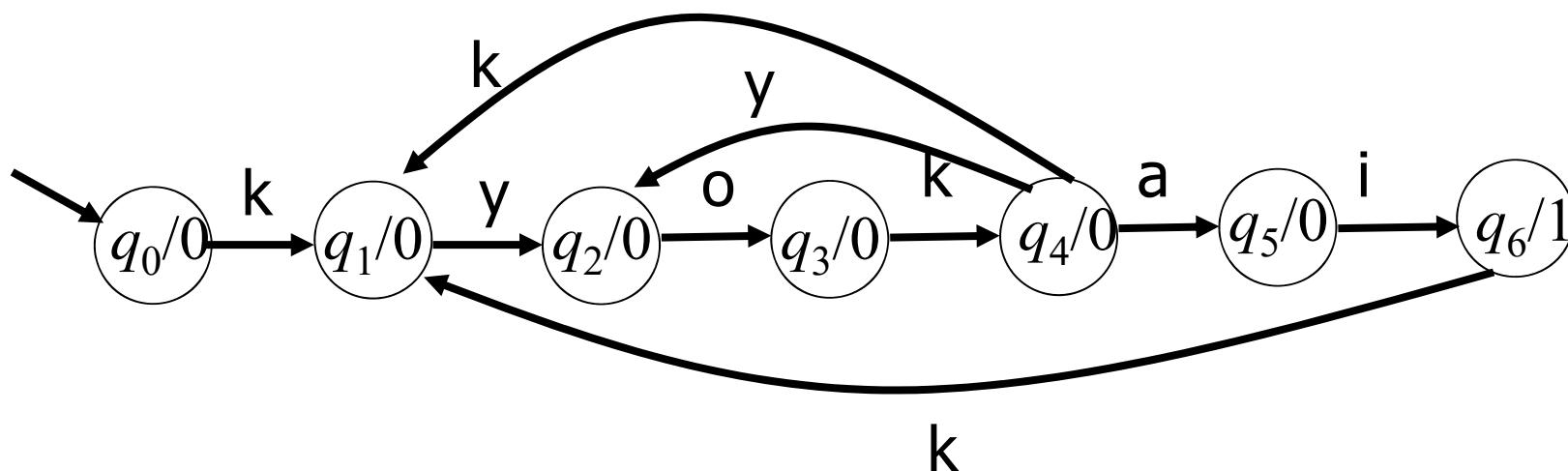


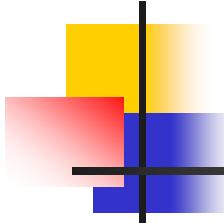
前綴り(prefix)から順序機械へ

キーワード $w = \text{kyokai}$

入力記号 c を加えたときの前綴りの前綴りと後綴りに注目

kyoky





複数個のキーワードの扱い

- キーワードが複数個ある場合に対しても同じ考え方
が適用可能

例 $s = \text{asukaratokyotokyotoni iku}$

$w_1 = \text{tokyo}$

$w_2 = \text{kyoto}$

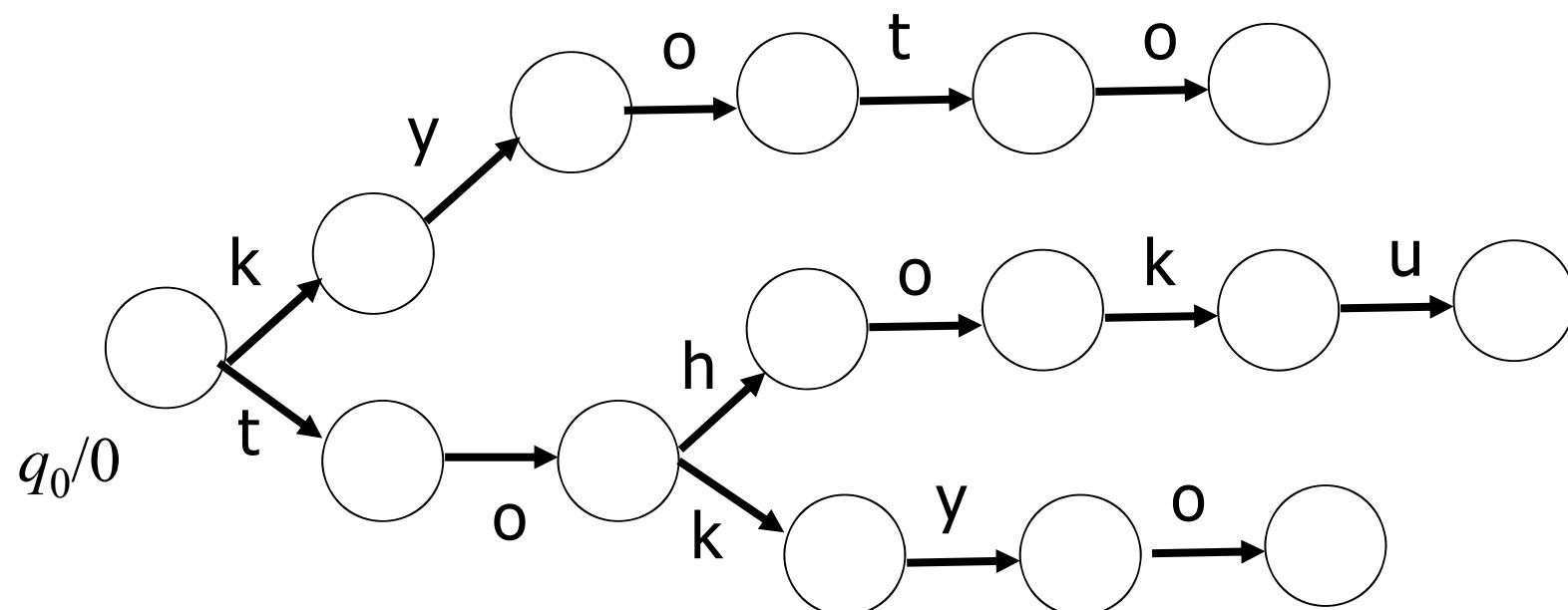
$\text{kyotoasudetokyotokyotoni iku}$

00002000000001020010200000

前綴木(prefix tree, trie)

- キーワードの前綴の重複を表現した木

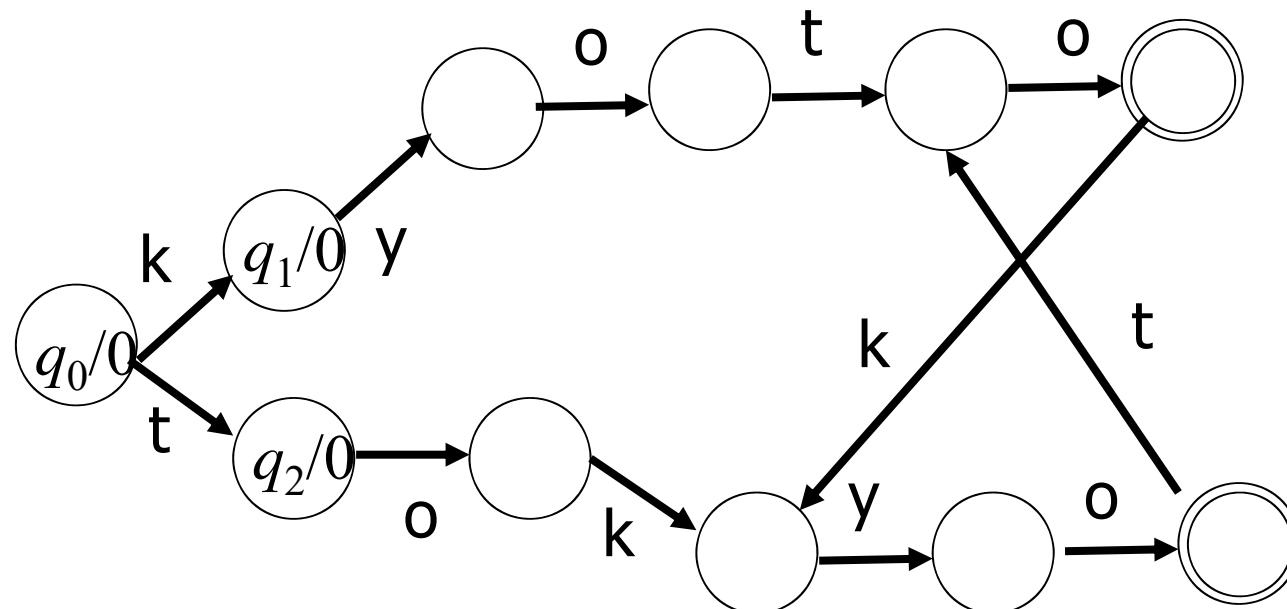
tokyo kyoto tohoku



前綴木からオートマトンへ

- 前綴木をDFAに変換する

tokyo kyoto

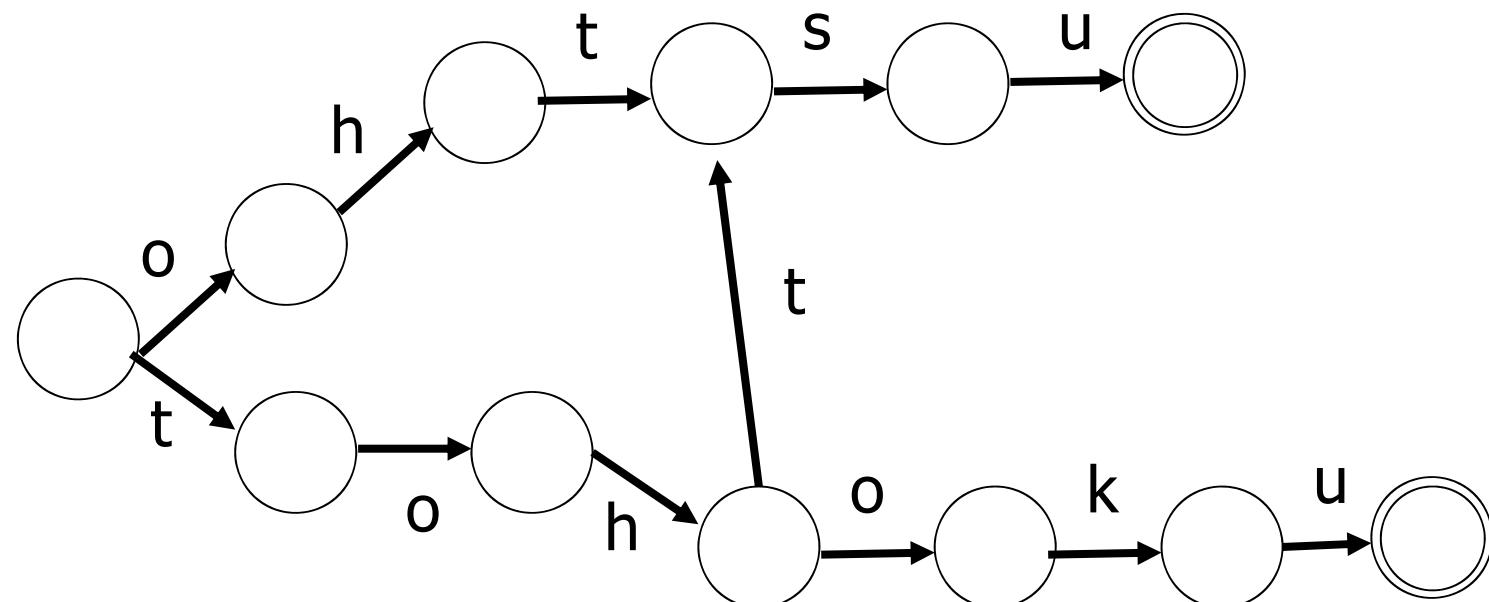


表記した遷移以外は $\delta(q,k)=q_1, \delta(q,t)=q_2$

前綴木からオートマトンへ

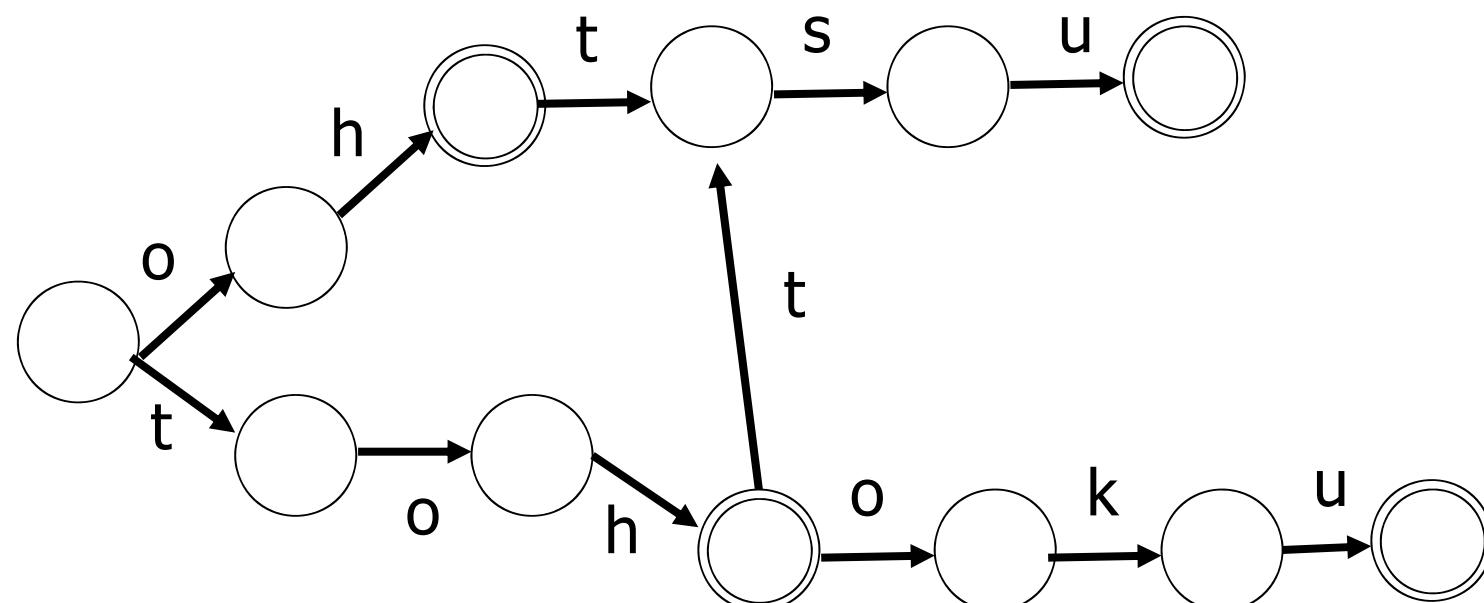
- 前綴木をDFAに変換する

tohoku ohtsu



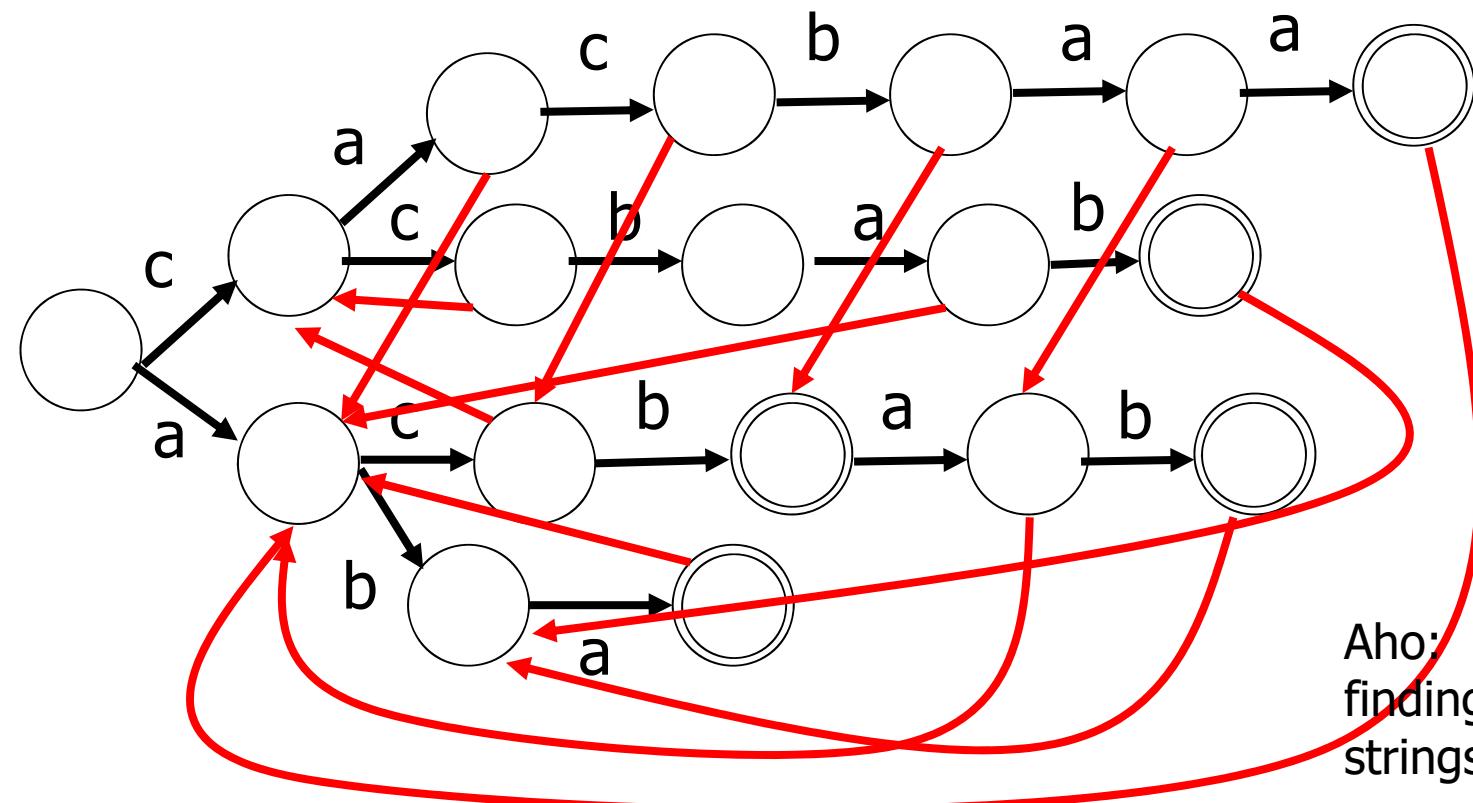
注1

- キーワード w_1 の一部であるようなキーワード w_2 に対しては、 w_2 に相当する状態の出力を1にしなければならない
tohoku ohtsu oh



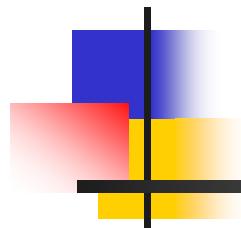
注2

- 失敗遷移(制限のついた ϵ 遷移)を用いると、オートマトンの構成の手間が少なくなることが知られている
(Aho-Corasick法)

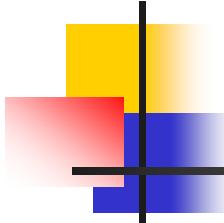


Aho: Algorithms for
finding patterns in
strings より引用

言語・オートマトン 文脈自由文法



山本章博
情報学研究科 知能情報学専攻
(工学部 情報学科)
2017年度版



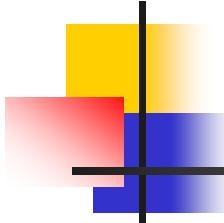
形式言語理論

言語を記号列・単語列の集合と抽象化した上で

- 文を生成(構成・定義)するための理論
 - 正しい記号列・単語列を生成(定義)する
- 文を受理するための理論
 - 送られてきた記号列・単語列が正しいかどうかを判定する
- 形式文法と受理機械



形式文法

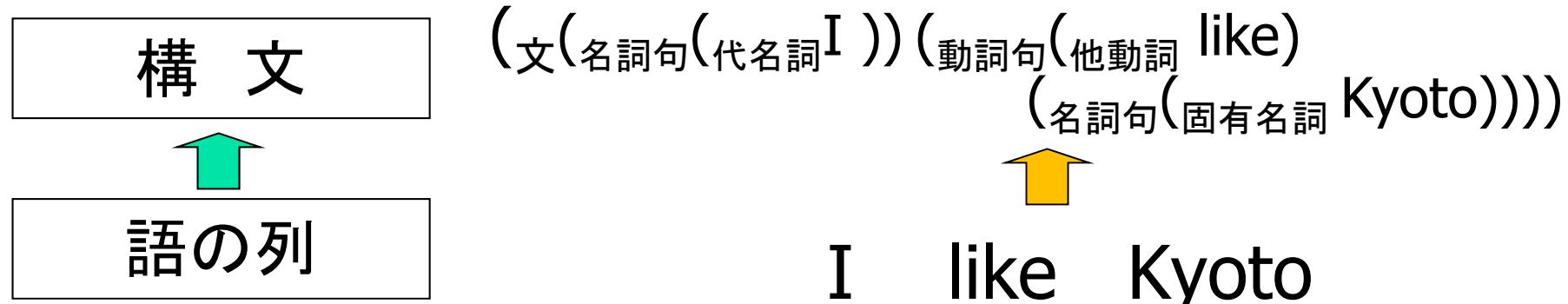


文法と構文

- 自然言語において、文を単語あるいは形態素、記号の列で構成するときには、ある規則に従っている。
- 規則を文法、文法に従った文が持つ構造を構文とよぶことにする

英語(自然言語)で記述された文

- 単語の品詞と文中での位置が構文を決める.
- 構文は 単語⇒句⇒節⇒文 のように階層を成す

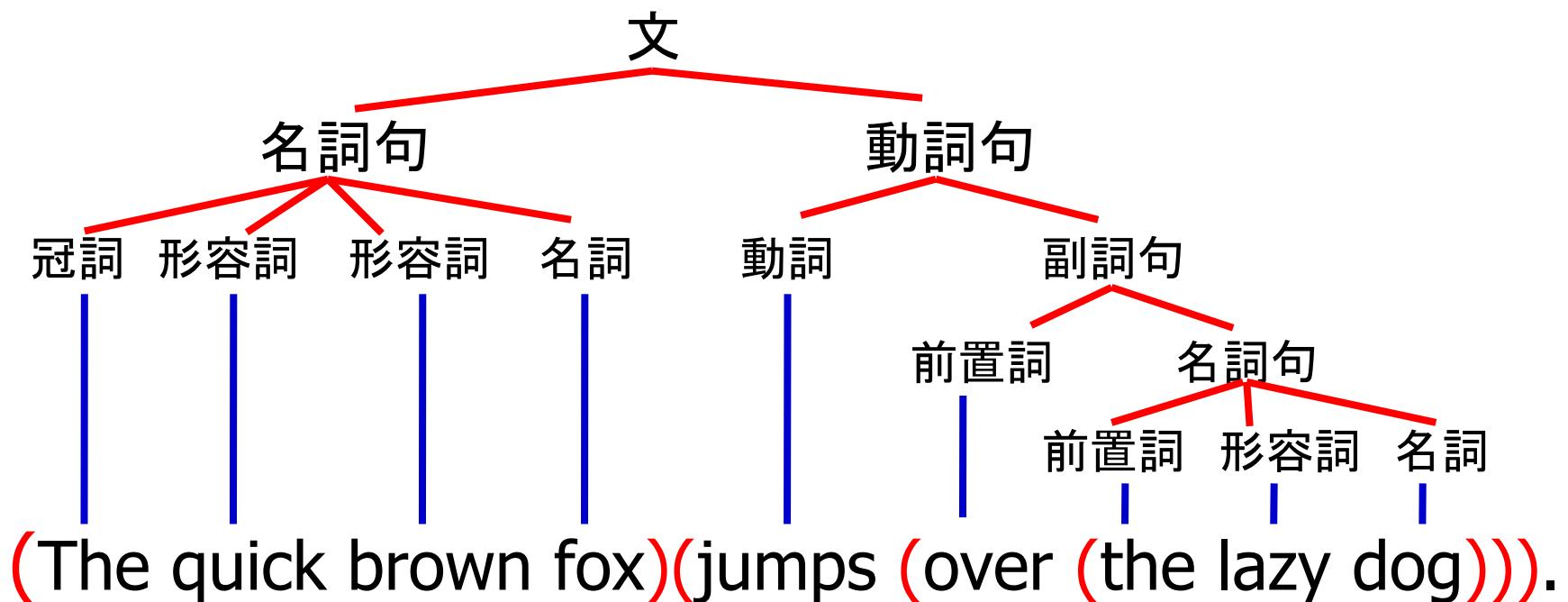


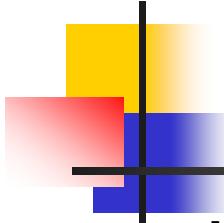
(The quick brown fox)(jumps (over (the lazy dog))).

冠詞 形容詞 形容詞 名詞 動詞 前置詞 冠詞 形容詞 名詞

英語(自然言語)で記述された文

- 単語の品詞と文中での位置が構文を決める.
- 構文は 単語⇒句⇒節⇒文 のように階層を成す





形式文法の考え方

- 文法を文を生成するための規則の集合ととらえる
 - $\alpha \rightarrow \beta$ という形の生成規則で表現する

文 → 名詞句 動詞句

名詞句 → 代名詞 動詞句 → 動詞

名詞句 → 固有名詞 動詞句 → 動詞 名詞句

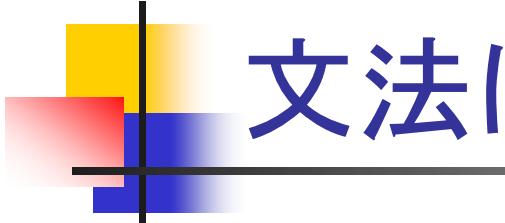
名詞句 → 冠詞 名詞 ...

...

代名詞 → I 代名詞 → you 代名詞 → he ...

固有名詞 → Kyoto 固有名詞 → Tom ...

動詞 → like 動詞 → have ...



文法による文の導出(1)

文 → 名詞句 動詞句 → 代名詞 動詞句
⇒ 固有名詞 動詞句 ⇒ I 動詞句 ⇒ I 動詞 名詞句
⇒ I like 名詞句 ⇒ I like 固有名詞 ⇒ I like Kyoto

文 → 名詞句 動詞句

名詞句 → 代名詞

動詞句 → 動詞

名詞句 → 固有名詞

動詞句 → 動詞 名詞句

名詞句 → 冠詞 名詞

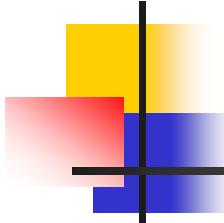
...

...

代名詞 → I 代名詞 → you 代名詞 → he ...

固有名詞 → Kyoto 固有名詞 → Tom ...

動詞 → like 動詞 → have ...



文法による文の導出(2)

文 ⇒ 名詞句 動詞句 ⇒ 代名詞 動詞句 ⇒ 固有名詞 動詞句
⇒ Tom 動詞句 ⇒ Tom 動詞 名詞句 ⇒ Tom have 名詞句
⇒ Tom have 冠詞 名詞 ⇒ Tom have a 名詞
⇒ Tom have a bag

文 → 名詞句 動詞句

名詞句 → 代名詞

動詞句 → 動詞

名詞句 → 固有名詞

動詞句 → 動詞 名詞句

名詞句 → 冠詞 名詞

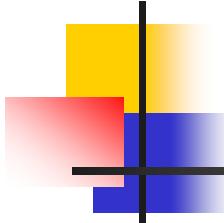
...

...

代名詞 → I 代名詞 → you 代名詞 → he ...

固有名詞 → Kyoto 固有名詞 → Tom ...

動詞 → like 動詞 → have ...



形式文法

- 形式文法 $G = (N, \Sigma, P, S)$

N : 非終端記号(構文要素)の有限集合

Σ : 終端記号(単語・記号)の有限集合

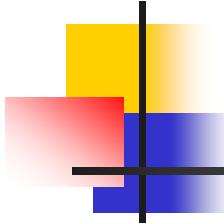
P : 生成規則の有限集合

$S \in N$: 開始記号

- **生成規則:** $\alpha \rightarrow \beta$ という形の規則

$\alpha \in (N \cup \Sigma)^+, \beta \in (N \cup \Sigma)^*$

- α, β は非終端記号(構文要素)と終端記号(単語・記号)の有限列

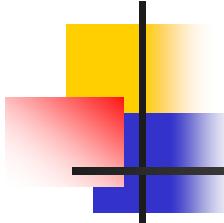


形式文法の例(1)

$G_1 = (N = \{S\}, \Sigma = \{a, b\}, P = \{S \rightarrow ab, S \rightarrow aSb\}, S)$

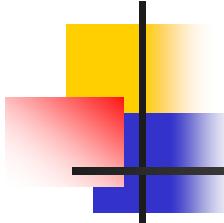
$G_2 = (N = \{S, A, B\}, \Sigma = \{a, b\},$
 $P = \{S \rightarrow aAB, A \rightarrow bBB, B \rightarrow abb\}, S)$

$G_3 = (N = \{S, A, B, C\}, \Sigma = \{a, b, c\},$
 $P = \{S \rightarrow aA, S \rightarrow bB, A \rightarrow aA, A \rightarrow bC,$
 $A \rightarrow b$
 $B \rightarrow aB, B \rightarrow bB, C \rightarrow aA, C \rightarrow bC,$
 $C \rightarrow b\}, S)$



形式文法の例(2)

$G_4 = (N = \{S, A, B, C, T\}, \Sigma = \{a, b\},$
 $P = \{S \rightarrow aBCT, S \rightarrow aBC,$
 $T \rightarrow ABCT, T \rightarrow ABC,$
 $BA \rightarrow AB, CA \rightarrow AC, CB \rightarrow BC,$
 $aA \rightarrow aa, aB \rightarrow ab, bB \rightarrow bb, bC \rightarrow bc,$
 $cC \rightarrow cc\}, S)$



形式文法の例(3)

$N = \{\text{名詞句}, \text{動詞句}, \text{代名詞}, \text{動詞}, \text{名詞句}, \text{固有名詞}, \text{冠詞}\}$

$\Sigma = \{I, you, he, \dots, Kyoto, Tom, \dots, like, have, \dots\}$

開始記号：文

$P = \{\text{文} \rightarrow \text{名詞句 } \text{動詞句}$

$\text{名詞句} \rightarrow \text{代名詞}$

$\text{動詞句} \rightarrow \text{動詞}$

$\text{名詞句} \rightarrow \text{固有名詞}$

$\text{動詞句} \rightarrow \text{動詞 } \text{名詞句}$

$\text{名詞句} \rightarrow \text{冠詞 } \text{名詞}$

...

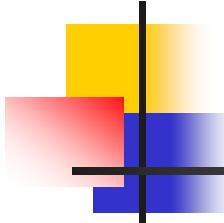
...

$\text{代名詞} \rightarrow I$ $\text{代名詞} \rightarrow you$ $\text{代名詞} \rightarrow he \dots$

$\text{固有名詞} \rightarrow Kyoto$ $\text{固有名詞} \rightarrow Tom \dots$

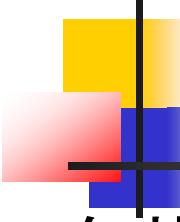
$\text{動詞} \rightarrow like$ $\text{動詞} \rightarrow have \dots$

}



形式文法の例(4)

$$G_5 = (N = \{S\}, \Sigma = \{ a, b, c, +, *, (,) \},$$
$$P = \{ S \rightarrow (S + S), S \rightarrow (S * S),$$
$$S \rightarrow a, S \rightarrow b, S \rightarrow c \}, S)$$



形式文法による導出

- 句構造文法 $G = (N, \Sigma, P, S)$

列 $\gamma\alpha\delta$ ($\gamma, \alpha, \delta \in (N \cup \Sigma)^*$) に対して生成規則 $\alpha \rightarrow \beta \in P$ が見つかれば、 $\gamma\alpha\delta \Rightarrow \gamma\beta\delta$ とかく。

- 列 $\gamma\beta\delta$ は $\gamma\alpha\delta$ から $\alpha \rightarrow \beta$ によって **直接に導出される** という。
- 一つの列に α が 2 箇所以上みつかっても、それらを **同時に** β に 書換えることはできない。

$$S * (S + S) \Rightarrow V * (S + S)$$

$$S * (S + S) \Rightarrow S * (V + S)$$

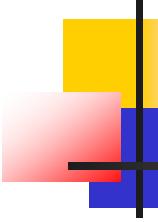
$$S * (S + S) \not\Rightarrow V * (V + V)$$

- 列 α と β は、 $\alpha = \beta$ であるか、または、

$$\alpha = \gamma_1 \Rightarrow \gamma_2 \Rightarrow \dots \Rightarrow \gamma_n = \beta$$

を満たす $\gamma_1, \gamma_2, \dots, \gamma_n$ が構成できるとき $\alpha \Rightarrow^* \beta$ とかく

- 列 β は α から G によって導出されるという。

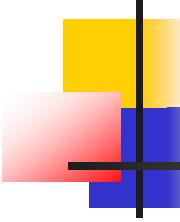


形式文法が生成する言語

- 句構造文法 $G = (N, \Sigma, P, S)$ に対して

$$L(G) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}$$

を G が **生成する言語** という



例(1)

$$G_1 = (N = \{S\}, \Sigma = \{a, b\}, P = \{S \rightarrow ab, S \rightarrow aSb\}, S)$$

$$S \Rightarrow ab$$

$$S \Rightarrow aSb \Rightarrow aabb$$

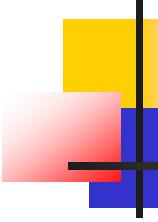
$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaabbb$$

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaaabbbb$$

なので

$$L(G_1) = \{w \in \{a, b\}^* \mid a^n b^n, n \in \mathbf{N}^+\}$$

■厳密には“証明”が必要



例(2)

$G_3 = (N = \{S, A, B, C\}, \Sigma = \{a, b\},$
 $P = \{S \rightarrow aA, S \rightarrow bB, A \rightarrow aA, A \rightarrow bC, A \rightarrow b$
 $B \rightarrow aB, B \rightarrow bB, C \rightarrow aA, C \rightarrow bC, C \rightarrow b\},$
 $S)$

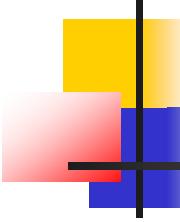
$$S \Rightarrow aA \Rightarrow aaA \Rightarrow aabC \Rightarrow aab$$

$$S \Rightarrow aA \Rightarrow aaA \Rightarrow aabC \Rightarrow aabb$$

$$S \Rightarrow aA \Rightarrow abC \Rightarrow abaA \Rightarrow abab$$

$$S \Rightarrow aA \Rightarrow abC \Rightarrow abbAC \Rightarrow abbaA \Rightarrow abbaaA$$

$$\Rightarrow abbaaaA \Rightarrow abbaaab$$

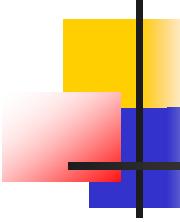


例(3)

$G_4 = (N = \{S, A, B, C, T\}, \Sigma = \{\text{a, b}\},$
 $P = \{S \rightarrow \text{aBCT}, S \rightarrow \text{aBC},$
 $T \rightarrow ABCT, T \rightarrow ABC,$
 $BA \rightarrow AB, CA \rightarrow AC, CB \rightarrow BC,$
 $\text{aA} \rightarrow \text{aa}, \text{aB} \rightarrow \text{ab}, \text{bB} \rightarrow \text{bb}, \text{bC} \rightarrow \text{bc},$
 $\text{cC} \rightarrow \text{cc}\}, S)$

$S \Rightarrow \text{aBC} \Rightarrow \text{abC} \Rightarrow \text{abc}$

$S \Rightarrow \text{aBCT} \Rightarrow \text{aBCABC} \Rightarrow \text{aBACBC} \Rightarrow \text{aBABCC}$
 $\Rightarrow \text{aABBCC} \Rightarrow \text{aaBBCC} \Rightarrow \text{aabBCC} \Rightarrow \text{aabbCC}$
 $\Rightarrow \text{aabbcC} \Rightarrow \text{aabbcc}$



例(4)

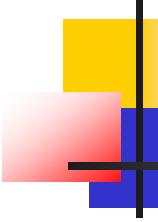
$$G_5 = (N = \{S\}, \Sigma = \{ a, b, c, +, *, (,) \},$$

$$P = \{ S \rightarrow (S + S), S \rightarrow (S * S),$$

$$S \rightarrow a, S \rightarrow b, S \rightarrow c \}, S)$$

$$S \Rightarrow (S + S) \Rightarrow (a + S) \Rightarrow (a + (S * S))$$

$$\Rightarrow (a + (b * S)) \Rightarrow (a + (b * c))$$



例(5)

文 ⇒ 名詞句 動詞句 ⇒ 代名詞 動詞句

⇒ 固有名詞 動詞句 ⇒ I 動詞句

⇒ I 動詞 名詞句 ⇒ I like 名詞句

⇒ I like 固有名 ⇒ I like Kyoto

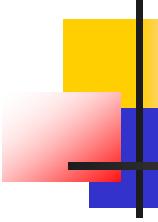
文 ⇒ 名詞句 動詞句 ⇒ 代名詞 動詞句

⇒ 固有名詞 動詞句 ⇒ Tom 動詞句

⇒ Tom 動詞 名詞句 ⇒ Tom have 名詞句

⇒ Tom have 冠詞 名詞 ⇒ Tom have a 名詞

⇒ Tom have a bag



例(5)

$G_3 = (N = \{S, A, B, C\}, \Sigma = \{a, b\},$

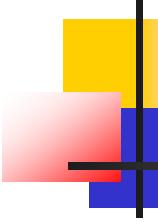
$P = \{S \rightarrow aA, S \rightarrow bB, A \rightarrow aA, A \rightarrow bC,$
 $B \rightarrow aB, B \rightarrow bB, C \rightarrow aA, C \rightarrow bC,$
 $C \rightarrow \varepsilon\}, S)$

$S \Rightarrow aA \Rightarrow aaA \Rightarrow aabC \Rightarrow aab$

$S \Rightarrow aA \Rightarrow aaA \Rightarrow aabC \Rightarrow aabbC \Rightarrow aabb$

$S \Rightarrow aA \Rightarrow abC \Rightarrow abaA \Rightarrow ababC \Rightarrow abab$

$S \Rightarrow aA \Rightarrow abC \Rightarrow abbAC \Rightarrow abbaA \Rightarrow abbaaaA$
 $\Rightarrow abbaaaA \Rightarrow abbaaabC \Rightarrow abbaaab$

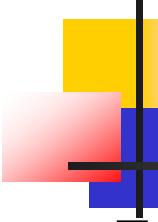


導出できないとは

$$G_1 = (\{S\}, \{a, b\}, \{S \rightarrow ab, S \rightarrow aSb\}, S)$$

$aabb \notin L(G_1) \Leftrightarrow S$ から $aabb$ が導出できない

- “導出できない”ということと
“導出できることを示す”ことは別
- “導出できることを示す”には
数学的に証明する
 - あらゆる導出の可能性を試みてそれらがことごとく失敗することを(プログラムを用いて)示す
 - などの方法がある

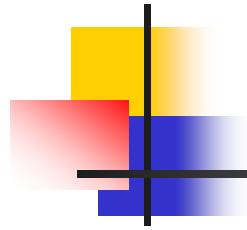


形式文法が定義する言語

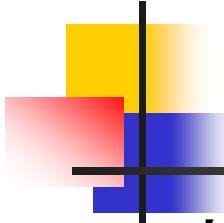
- 形式文法 G , 語 $w \in \Sigma^*$
- $S \Rightarrow^* w$ となる導出が構成できるとき, w は G によって **導出される** または **生成される** という.
- G が生成する言語 $L(G) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}$
- The language defined a context free grammar $G = (N, \Sigma, P, S)$ is $L(G) = \{w \in \Sigma^* \mid \text{there is a derivation } S \Rightarrow^* w\}.$

例

- $\Sigma = \{a, b\}$ $S \rightarrow ab, S \rightarrow aSb$
- $L = \{a^n b^n \mid n \geq 1\} = \{\underbrace{a \dots a}_{n \text{ times}} \underbrace{b \dots b}_{n \text{ times}} \mid n \geq 1\}$ $= \{ab, aabb, aaabbb, aaaabbbb, \dots\}$



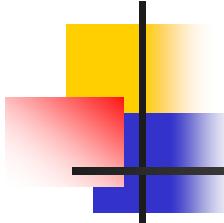
文脈自由文法と導出木



文脈依存文法と文脈自由文法

ややこしい(α, β includes empty strings)

- 文法 $G = (N, \Sigma, P, S)$ は、 P 中の全ての生成規則が $\alpha A \beta \rightarrow \alpha \gamma \beta$ ($A \in N, \alpha, \beta, \gamma \in (N \cup \Sigma)^*$)
という形であるとき文脈依存 (context sensitive) 文法という
- 文法 $G = (N, \Sigma, P, S)$ は、 P 中の全ての生成規則が $A \rightarrow \gamma$ ($A \in N, \gamma \in (N \cup \Sigma)^*$)
という形であるとき文脈自由 (context free) 文法という

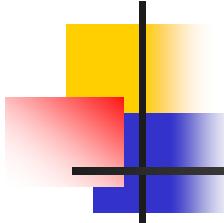


文脈自由文法の例

$G_1 = (N = \{S\}, \Sigma = \{a, b\}, P = \{S \rightarrow ab, S \rightarrow aSb\}, S)$

$G_2 = (N = \{S, A, B\}, \Sigma = \{a, b\},$
 $P = \{S \rightarrow aAB, A \rightarrow bBB, B \rightarrow abb\}, S)$

$G_3 = (N = \{S, A, B, C\}, \Sigma = \{a, b\},$
 $P = \{S \rightarrow aA, S \rightarrow bB, A \rightarrow aA, A \rightarrow bC,$
 $A \rightarrow b,$
 $B \rightarrow aB, B \rightarrow bB, C \rightarrow aA, C \rightarrow bC,$
 $C \rightarrow b\}, S)$



文脈依存文法の例

$G_4 = (N = \{S, A, B, C, T\}, \Sigma = \{a, b\},$

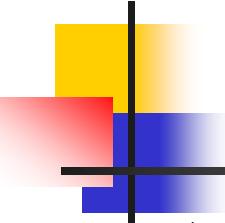
$P = \{S \rightarrow aBCT, S \rightarrow aBC,$

$T \rightarrow ABCT, T \rightarrow ABC,$

$BA \xrightarrow{*} AB, CA \xrightarrow{*} AC, CB \xrightarrow{*} BC,$

$aA \rightarrow aa, aB \rightarrow ab, bB \rightarrow bb, bC \rightarrow bc,$

$cC \rightarrow cc\}, S)$



BNF(Backus-Naur form)

- 主にプログラミング言語設計の分野では、文脈自由文法の規則において
 - 非終端記号を<記号列>で表わし
 - 終端記号を"記号列"で表わし
 - の代わりに ::= を用いている。また左辺が同じ規則の右辺をまとめて | で区切って表す

例 生成規則の集合

{式 → (式 + 式), 式 → (式 * 式), 式 → a, 式 → b, 式 → c}

を表すBNF

<式> ::= " (" <式>" +" <式>")" | " (" <式>" +" <式>")"

| "a" | "b" | "a"

構文木(導出木)

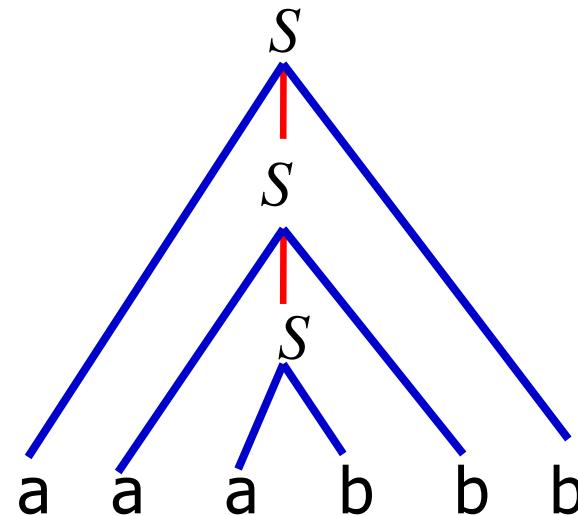
- 文脈自由文法の導出は木構造を用いて表現できる。
 - 文 w の導出を表す木を導出木あるいは**構文木**(*parsing tree*)という

例 $G_1 = (N = \{S\}, \Sigma = \{a, b\}, P = \{S \rightarrow ab, S \rightarrow aSb\}, S)$

$$S \Rightarrow aSb$$

$$\Rightarrow aaSbb$$

$$\Rightarrow aaabb$$



導出木の例(2)

$$N = \{ S \}$$

$$\Sigma = \{ a, b, c, +, *, (,) \}$$

開始記号 : S

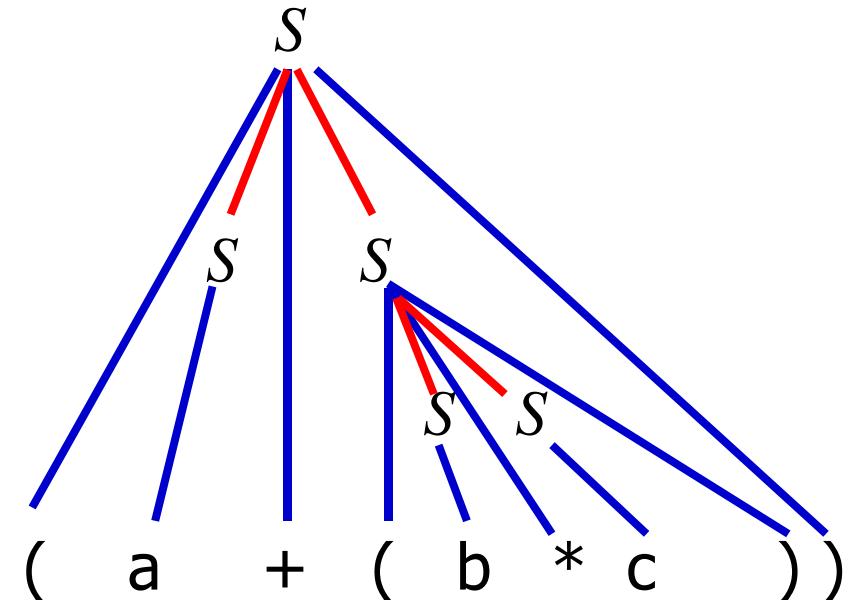
$$P = \{ S \rightarrow (S + S),$$

$$S \rightarrow (S * S),$$

$$S \rightarrow a,$$

$$S \rightarrow b,$$

$$S \rightarrow c \}$$

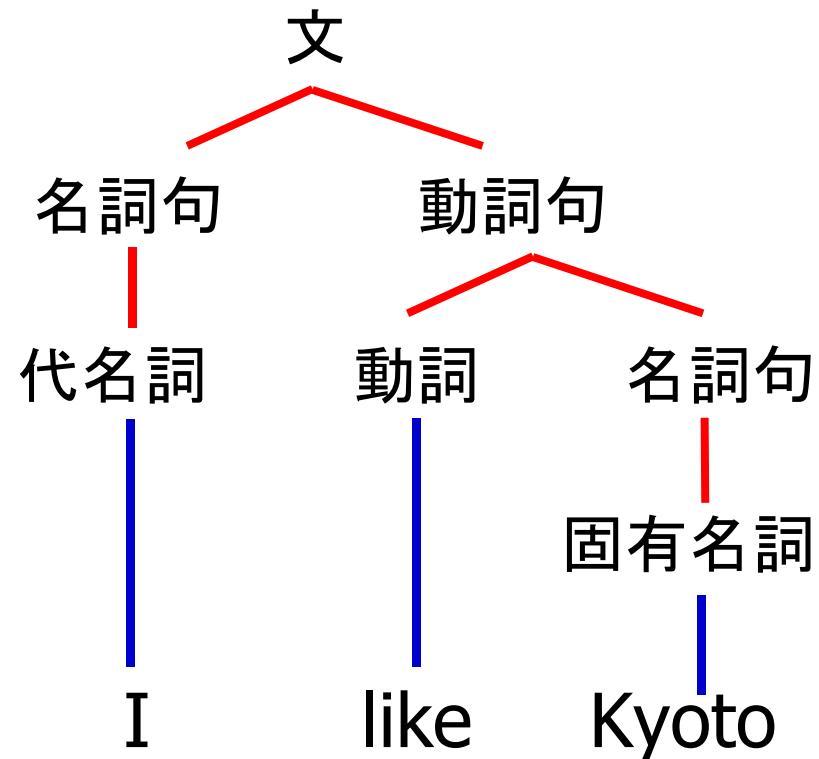


$$S \Rightarrow (S + S) \Rightarrow (a + S) \Rightarrow (a + (S * S))$$

$$\Rightarrow (a + (b * S)) \Rightarrow (a + (b * c))$$

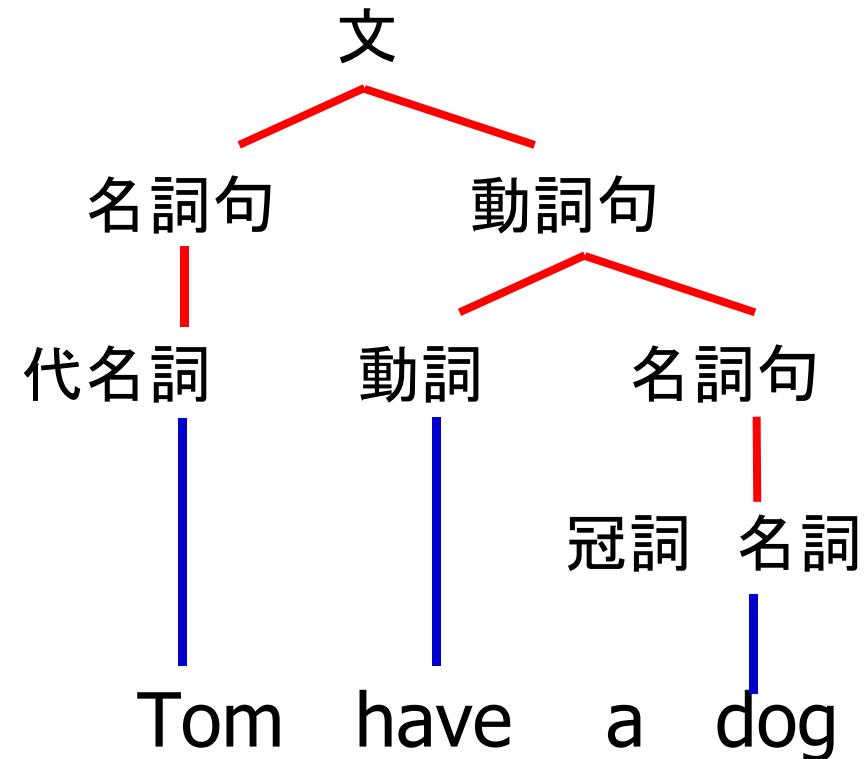
導出木の例(3)

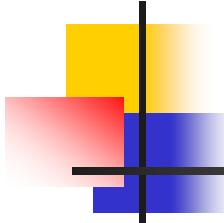
文 ⇒ 名詞句 動詞句
⇒ 代名詞 動詞句
⇒ 固有名詞 動詞句
⇒ I 動詞句
⇒ I 動詞 名詞句
⇒ I like 名詞句
⇒ I like 固有名詞
⇒ I like Kyoto



導出木の例(4)

- 文 ⇒ 名詞句 動詞句
⇒ 代名詞 動詞句
⇒ 固有名詞 動詞句
⇒ Tom 動詞句
⇒ Tom 動詞 名詞句
⇒ Tom have 名詞句
⇒ Tom have 冠詞 名詞
⇒ Tom have a 名詞
⇒ Tom have a bag



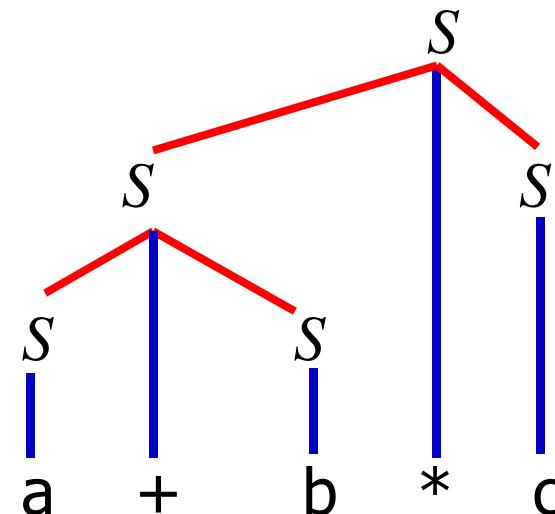
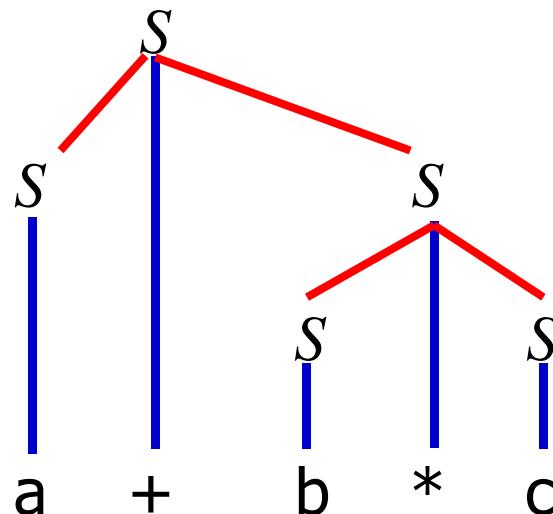


構文解析(Parsing)

- 単語(記号)の列 σ が与えられたとき, その構文を決定すること
- 
- 単語(記号)の列 σ が与えられたとき, “文”から始まり σ で終わる導出を構成する.
 - 構文解析の結果得られる構文を表す木を**構文木**(parse tree)あるいは**導出木**(derivation tree)とよぶ

曖昧な文法(1)

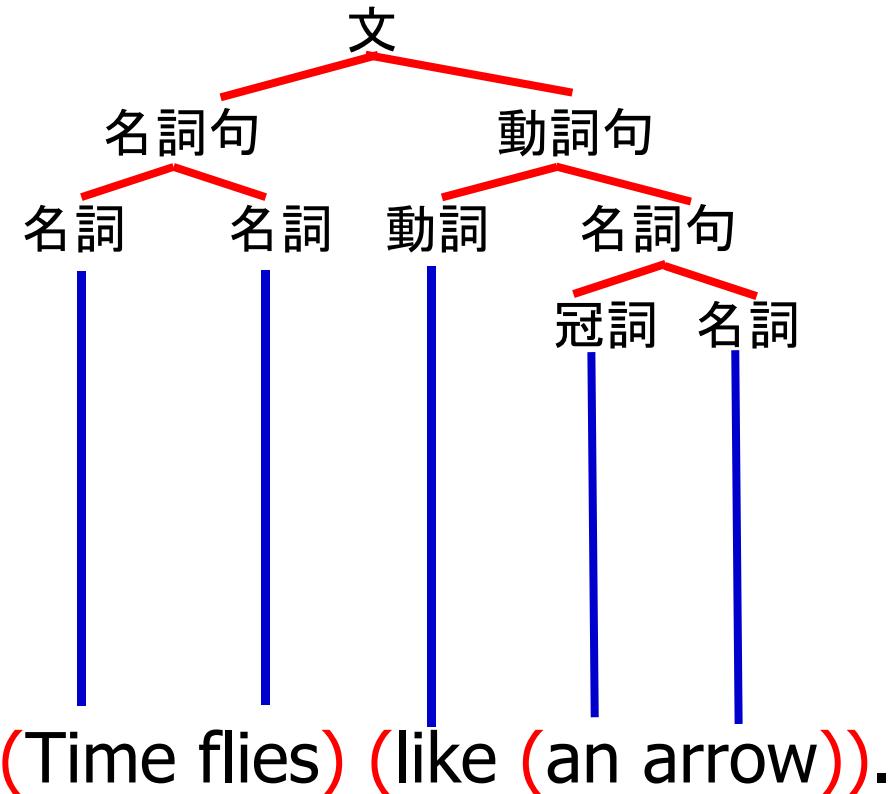
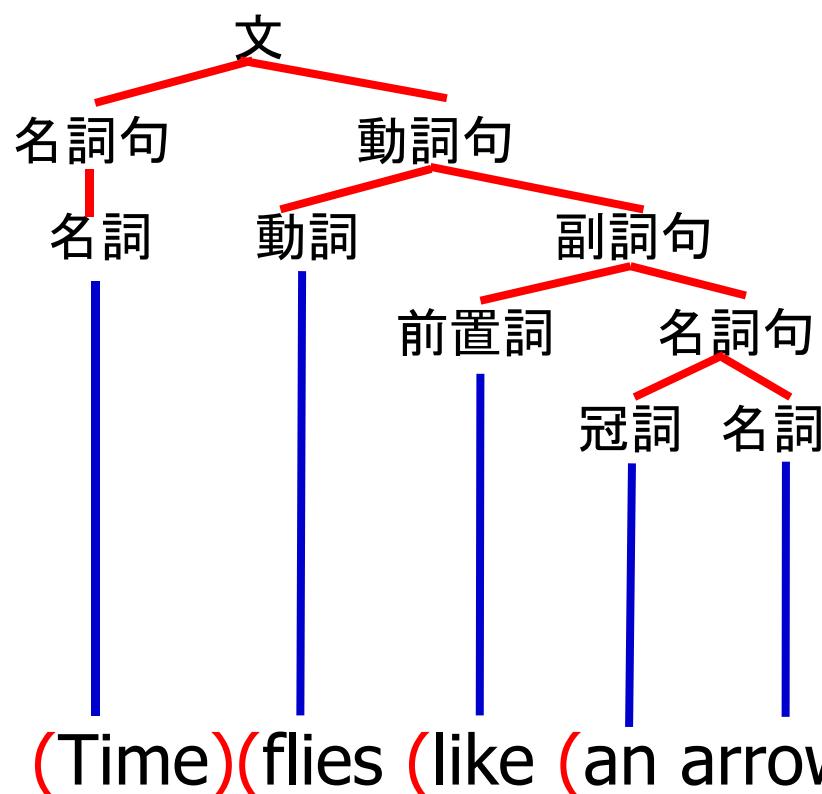
- ある文法を用いると、一つの単語(記号)の列 σ に対して異なる2種類以上の導出が構成されることがあるとき、その文法は**曖昧である**という。

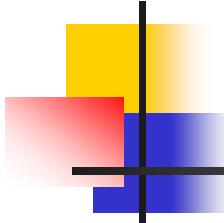


曖昧な文法(2)

- 名詞句 → 名詞
- 名詞句 → 名詞 名詞

動詞 → flies 動詞 → like
名詞 → flies 前置詞 → like



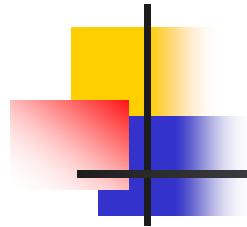


本質的に曖昧な言語

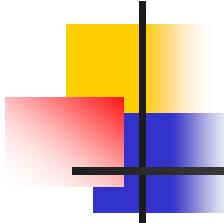
- 文脈自由言語 L をどのような文脈自由文法 G で表現しても G が曖昧であるとき, L は**本質的に曖昧である**という.

例 $L = \{a^i b^i c^j \mid i \in \mathbb{N}^+, j \in \mathbb{N}^+\}$

- 文脈自由文法 G を**任意に**与えたとき, $L(G)$ が本質的に曖昧であるかどうかを判定するアルゴリズムは存在しない.



構文解析



構文解析(Parsing)

- 単語(記号)の列 σ が与えられたとき, その構文を決定すること
- 
- 単語(記号)の列 σ が与えられたとき, “文”から始まり σ で終わる導出を構成する.
 - 構文解析の結果得られる構文を表す木を**構文木**(parse tree)あるいは**導出木**(derivation tree)とよぶ

構文木(導出木)

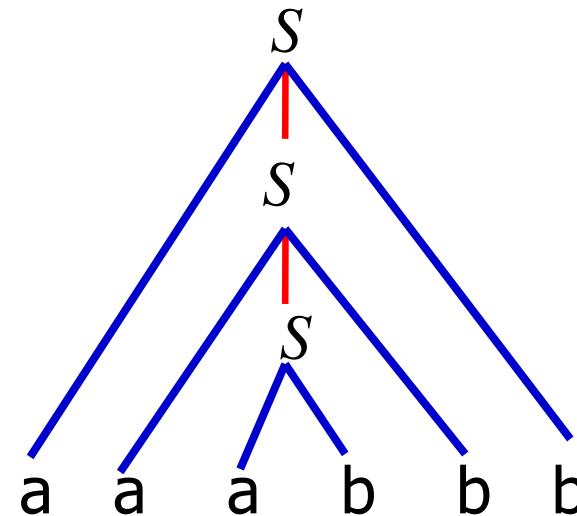
- 文脈自由文法の導出は木構造を用いて表現できる。
 - 文 w の導出を表す木を導出木あるいは**構文木**(*parsing tree*)という

例 $G_1 = (N = \{S\}, \Sigma = \{a, b\}, P = \{S \rightarrow ab, S \rightarrow aSb\}, S)$

$$S \Rightarrow aSb$$

$$\Rightarrow aaSbb$$

$$\Rightarrow aaabb$$



導出木の例(2)

$N = \{ \text{式} \}$

$\Sigma = \{ a, b, c, +, *, (,) \}$

開始記号 : 式

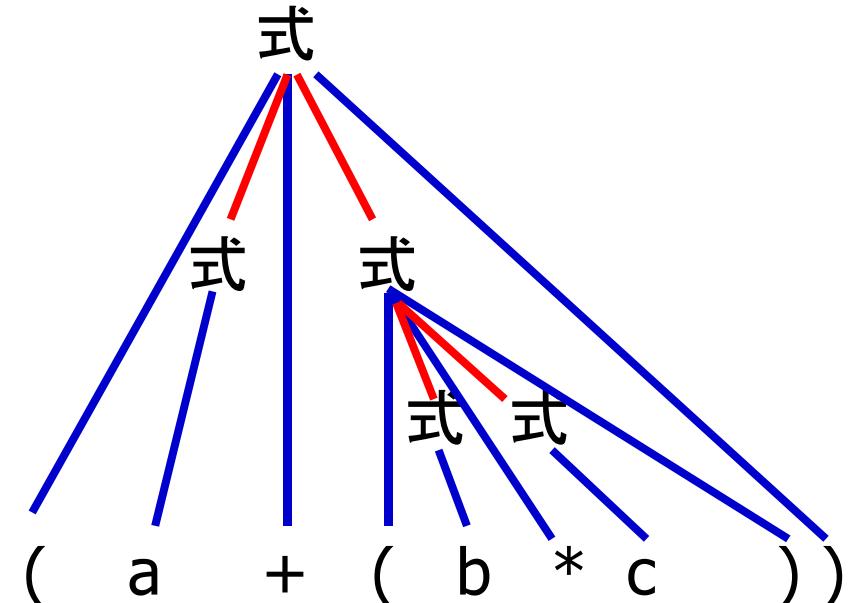
$P = \{ \text{式} \rightarrow (\text{式} + \text{式}),$

$\text{式} \rightarrow (\text{式} * \text{式}),$

$\text{式} \rightarrow a,$

$\text{式} \rightarrow b,$

$\text{式} \rightarrow c \}$



$\text{式} \Rightarrow (\text{式} + \text{式}) \Rightarrow (a + \text{式}) \Rightarrow (a + (\text{式} * \text{式}))$

$\Rightarrow (a + (b * \text{式})) \Rightarrow (a + (b * c))$

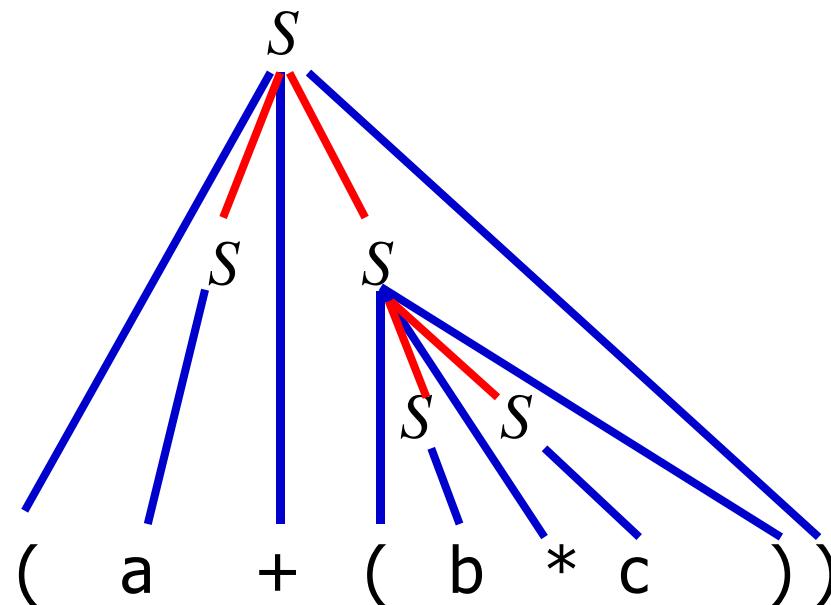
導出木の例(2)

$$G_4 = (N = \{S\}, \Sigma = \{ a, b, c, +, *, (,) \}, P, S)$$

$$P = \{S \rightarrow (S+S), S \rightarrow (S^*S), S \rightarrow a, S \rightarrow b, S \rightarrow c\}$$

$$S \Rightarrow (S + S) \Rightarrow (a + S) \Rightarrow (a + (S * S))$$

$$\Rightarrow (a + (b * S)) \Rightarrow (a + (b * c))$$



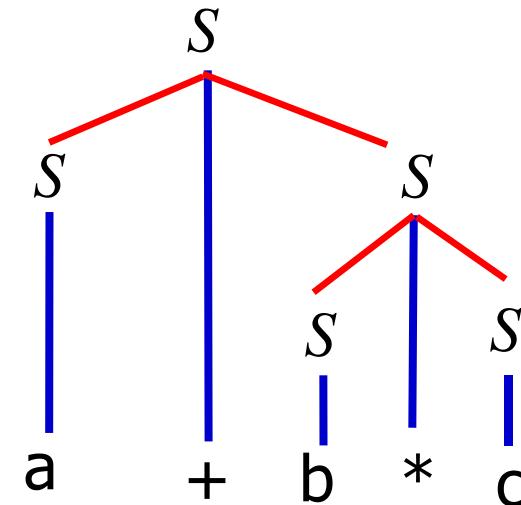
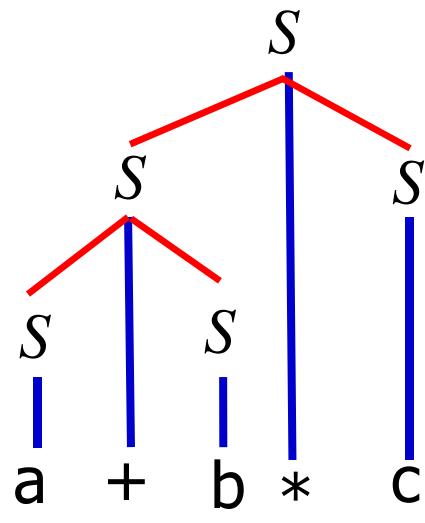
導出木の例(3)

$$G_5 = (N = \{S\}, \Sigma = \{ a, b, c, +, * \}, P, S)$$

$$P = \{S \rightarrow S + S, S \rightarrow S * S, S \rightarrow a, S \rightarrow b, S \rightarrow c\}$$

$$S \Rightarrow S + S \Rightarrow a + S \Rightarrow a + S * S \Rightarrow a + b * S \Rightarrow a + b * c$$

$$S \Rightarrow S * S \Rightarrow S * c \Rightarrow S + S * c \Rightarrow S + b * c \Rightarrow a + b * c$$

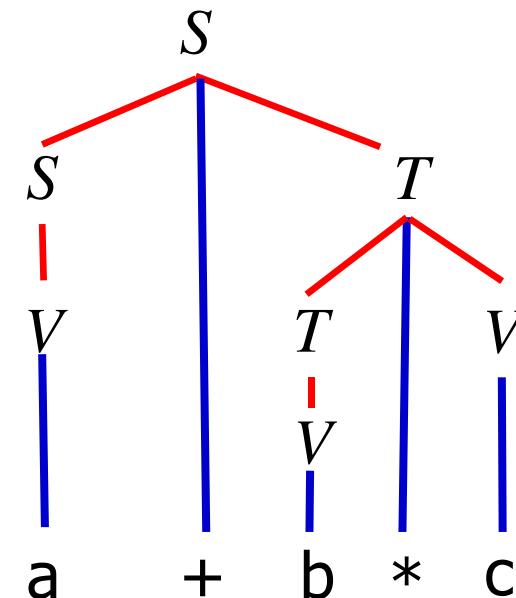


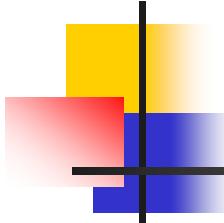
導出木の例(4)

$$G_6 = (N = \{S\}, \Sigma = \{ a, b, c, +, * \}, P, S)$$

$$\begin{aligned}P = & \{ S \rightarrow S + T, S \rightarrow T, S \rightarrow V, T \rightarrow T * V, T \rightarrow V \\& V \rightarrow a, V \rightarrow b, V \rightarrow c \}\end{aligned}$$

$$\begin{aligned}S &\Rightarrow S + T \Rightarrow V + T \Rightarrow a + T \Rightarrow a + T * V \Rightarrow a + V * V \\&\Rightarrow a + b * V \Rightarrow a + b * c\end{aligned}$$





Chomsky標準形の文法

- 文脈自由文法 $G = (N, \Sigma, P, S)$ は、 P 中の全ての生成規則が次の3種類のいずれかであるとき, Chomsky標準形であるという.

$$A \rightarrow BC \quad A \rightarrow c \quad S \rightarrow \varepsilon$$

ただし $A, B, C \in N, c \in \Sigma$

例 $G_1' = (\{S, A, B\}, \{a, b\}, P, S)$

$$P = \{S \rightarrow AT, T \rightarrow SB, A \rightarrow a, B \rightarrow b\}$$

$$L(G_1') = \{a^n b^n \mid n \in \mathbf{N}^+\}$$

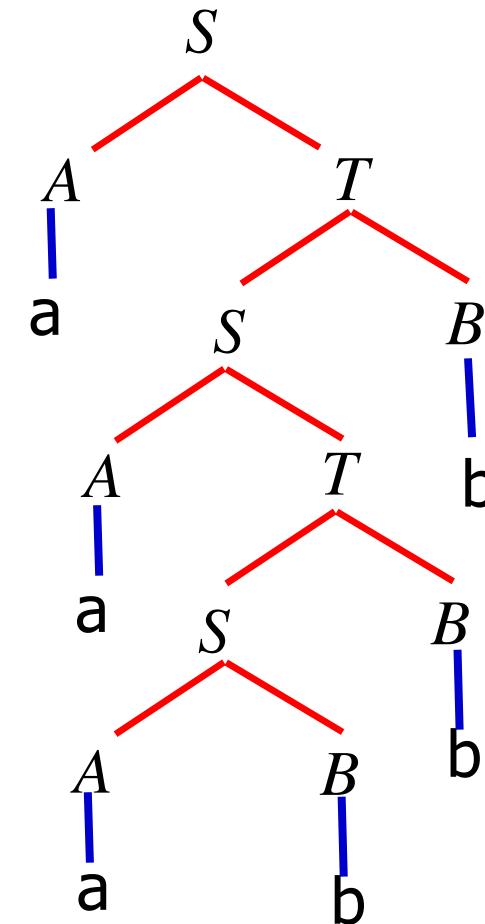
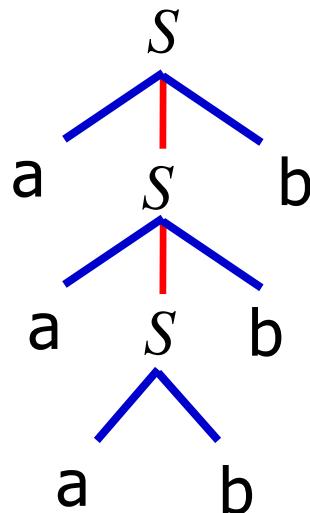
$$= L(G_1)$$

$$\text{ここで } G_1 = (\{S\}, \{a, b\}, \{S \rightarrow aSb, S \rightarrow ab\}, S)$$

Chomsky標準形と構文木

$$P = \{S \rightarrow ab, S \rightarrow aSb\}, S)$$

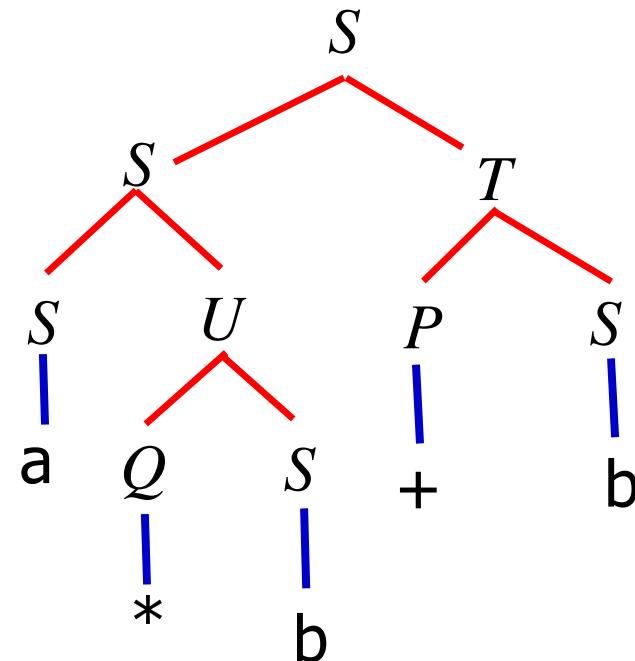
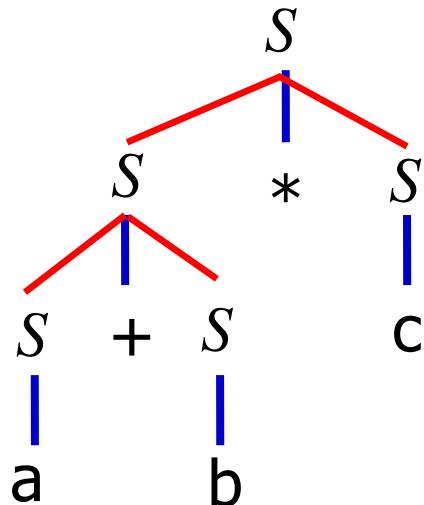
$$P' = \{S \rightarrow AT, S \rightarrow AB, T \rightarrow SB, A \rightarrow a, B \rightarrow b\}$$

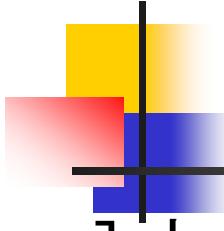


Chomsky標準形と構文木

$$P = \{S \rightarrow S+S, S \rightarrow S^*S, S \rightarrow a, S \rightarrow b, S \rightarrow c\}$$

$$\begin{aligned} P' = & \{S \rightarrow ST, T \rightarrow PS, S \rightarrow SU, U \rightarrow QS, \\ & P \rightarrow +, Q \rightarrow *, S \rightarrow a, S \rightarrow b, S \rightarrow c\} \end{aligned}$$





Cocke-Kasami-Younger法

入力:

Chomsky標準形の文脈自由文法 $G = (N, \Sigma, P, S)$

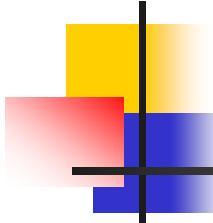
記号列 $w = c_1c_2\dots c_n$

手続き

- 動的計画法による上昇型構文解析
- 解析行列 $A[i, j]$ を順次埋めてゆく
- $A[i, j]$ には長さ i の c_j ら始まるの分記号列 $c_jc_{j+1}\dots c_{j+i-1}$ を導出可能な非終端記号 A がすべて格納される

$$A \Rightarrow BC \Rightarrow \dots \Rightarrow c_jc_{j+1}\dots c_{j+i-1}$$

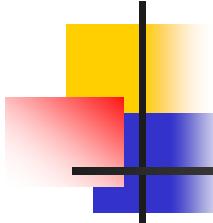
1. $A[1, j]$ には記号 c_j を直接に導出する非終端記号を格納する
2. $A[i, j]$ には、すべての $k=1, \dots, i-1$ について、 $B \in A[i-k, j]$ と $C \in A[k, j+k]$ の各組み合わせについて、 BC を直接導出するような A をすべて格納する



CKYの実行例(1)

- $P' = \{S \rightarrow AT, S \rightarrow AB, T \rightarrow SB, A \rightarrow a, B \rightarrow b\}$
- $w = a \ a \ a \ b \ b \ b$
 $c_1 \ c_2 \ c_3 \ c_4 \ c_5 \ c_6$

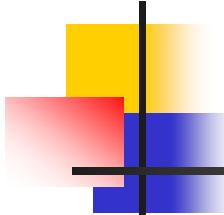
	1	2	3	4	5	6
	a	a	a	b	b	b
1	A	A	A	B	B	B
2	\emptyset	\emptyset	S	\emptyset	\emptyset	
3	\emptyset	\emptyset	T	\emptyset		
4	\emptyset	S	\emptyset			
5	\emptyset	T				
6	S					



CKYの実行例(2)

- $P' = \{S \rightarrow ST, T \rightarrow PS, S \rightarrow SU, U \rightarrow QS,$
 $P \rightarrow +, Q \rightarrow *, S \rightarrow c, S \rightarrow a, S \rightarrow b, S \rightarrow c\}$
- $w = a^* b + c$
 $c_1 c_2 c_3 c_4 c_5$

	1	2	3	4	5
	a	*	b	+	c
2	S	Q	S	P	S
3	\emptyset	U	\emptyset	T	
4	S	\emptyset	S		
5	\emptyset	U			
6	$S(S, S)$				



単位生成規則の除去

- 文脈自由文法 $G = (N, \Sigma, P, S)$ における単位生成規則：
 $A \rightarrow B$ ($A, B \in N$)
 - 単位生成規則は“非終端記号の付け替え”である.
- N 中のすべての非終端記号 A について

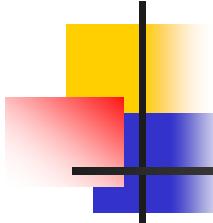
$$A \Rightarrow B_1 \Rightarrow B_2 \Rightarrow \dots \Rightarrow B_n \Rightarrow \gamma$$

(A, B_1, B_2, \dots, B_n は互いに異なる非終端記号

$$\gamma \in (N \cup \Sigma)^* - N)$$

という形の導出をすべて求め, $A \rightarrow \gamma$ を P に追加する.

- 得られた P から単位生成規則を取り除いた結果を P' とする
- $G' = (N, \Sigma, P', S)$ とすれば, $L(G) = L(G')$



Chomsky標準形の文法への変換

- 単位生成規則を含まない文脈自由文法 $G = (N, \Sigma, P, S)$
 P 中の各規則 $A \rightarrow \gamma$ に次の操作を適用する
 - すでに $A \rightarrow \gamma$ がChomsky標準形のいずれかの形になつていれば何もしない
 - $\gamma = cB$ ($c \in \Sigma, B \in N$)のとき, $A \rightarrow cB$ を P から除いて
 $A \rightarrow [c]B$ と $[c] \rightarrow c$ を加える. ただし, $[c]$ は記号 c に対して新たに生成する非終端記号とする.
 - $\gamma = Bc$ ($c \in \Sigma, B \in N$)のとき, $A \rightarrow cB$ を P から除いて
 $A \rightarrow B[c]$ と $[c] \rightarrow c$ を加える.
 - $\gamma = cd$ ($c, d \in \Sigma$)のとき, $A \rightarrow cd$ を P から除いて
 $A \rightarrow [c][d]$ と $[c] \rightarrow c, [d] \rightarrow d$ を加える.



Chomsky標準形の文法への変換(続)

5. $\gamma = X_1 X_2 \dots X_n$ ($n \geq 3$, $X_i \in N \cup \Sigma$) のとき, $A \rightarrow \gamma$ を P から除き, 以下の規則をすべて加える.

$$A \rightarrow [X_1][X_2 X_3 \dots X_n]$$

$$[X_2 X_3 \dots X_n] \rightarrow [X_2][X_3 \dots X_n]$$

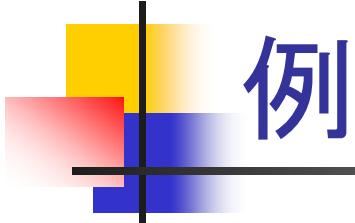
...

$$[X_{n-1} X_n] \rightarrow [X_{n-1}][X_n]$$

ただし, $X_i = c \in \Sigma$ のとき $[X_i] = [c]$ は上と同様であり, $[c] \rightarrow c$ も追加する

$$X_i = B \in N \text{ のとき } [X_i] = B$$

$[X_i \dots X_n]$ ($2 \leq i \leq n-1$) は $X_i \dots X_n$ に対する新しい非終端記号とする.



例

$$G_1 = (\{S\}, \{a, b\}, \{S \rightarrow aSb, S \rightarrow ab\}, S)$$

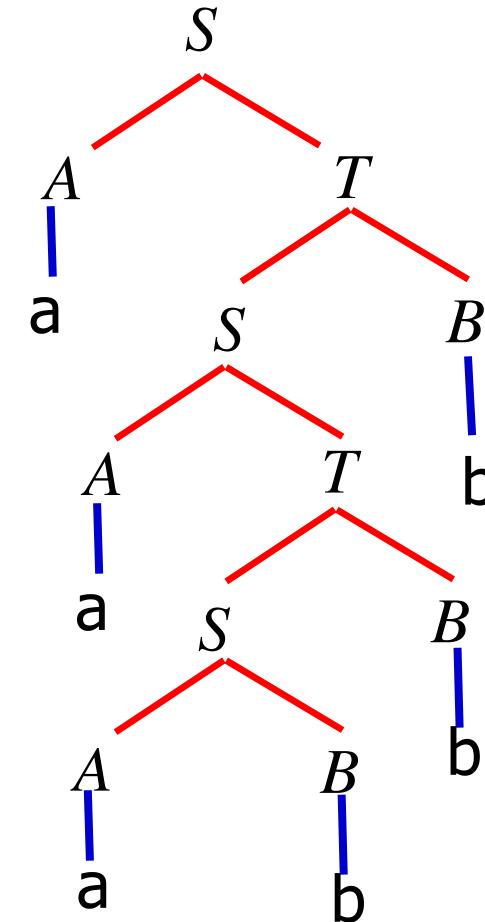
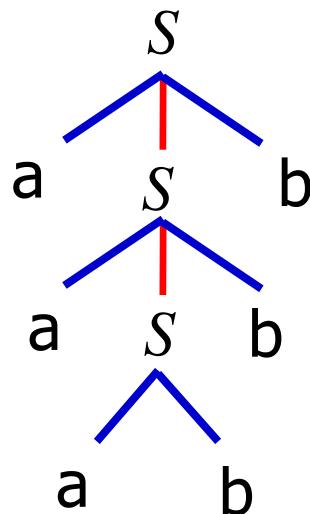
- $S \rightarrow aSb$ に 5 を適用すると, $S \rightarrow aSb$ を除いて
 $S \rightarrow [a][Sb], [Sb] \rightarrow S[b], [a] \rightarrow a, [b] \rightarrow b$ を加える
- $S \rightarrow ab$ に 4 を適用すると, $S \rightarrow ab$ を除いて
 $S \rightarrow [a][b]$ を加える
- (人間にとて)読みやすいように, $[a], [b], [Sb]$ をそれぞれ A, B, T と表せば,

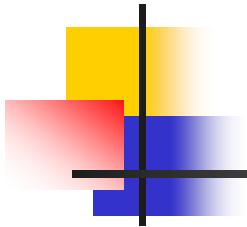
$$G_1' = (\{S, T, A, B\}, \{a, b\}, \\ \{S \rightarrow AT, S \rightarrow AB, T \rightarrow SB, A \rightarrow a, B \rightarrow b\}, S)$$

Chomsky標準形と構文木

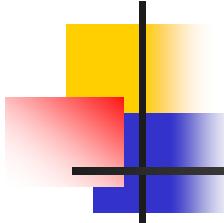
$$P = \{S \rightarrow ab, S \rightarrow aSb\}, S)$$

$$P' = \{S \rightarrow AT, S \rightarrow AB, T \rightarrow SB, A \rightarrow a, B \rightarrow b\}$$





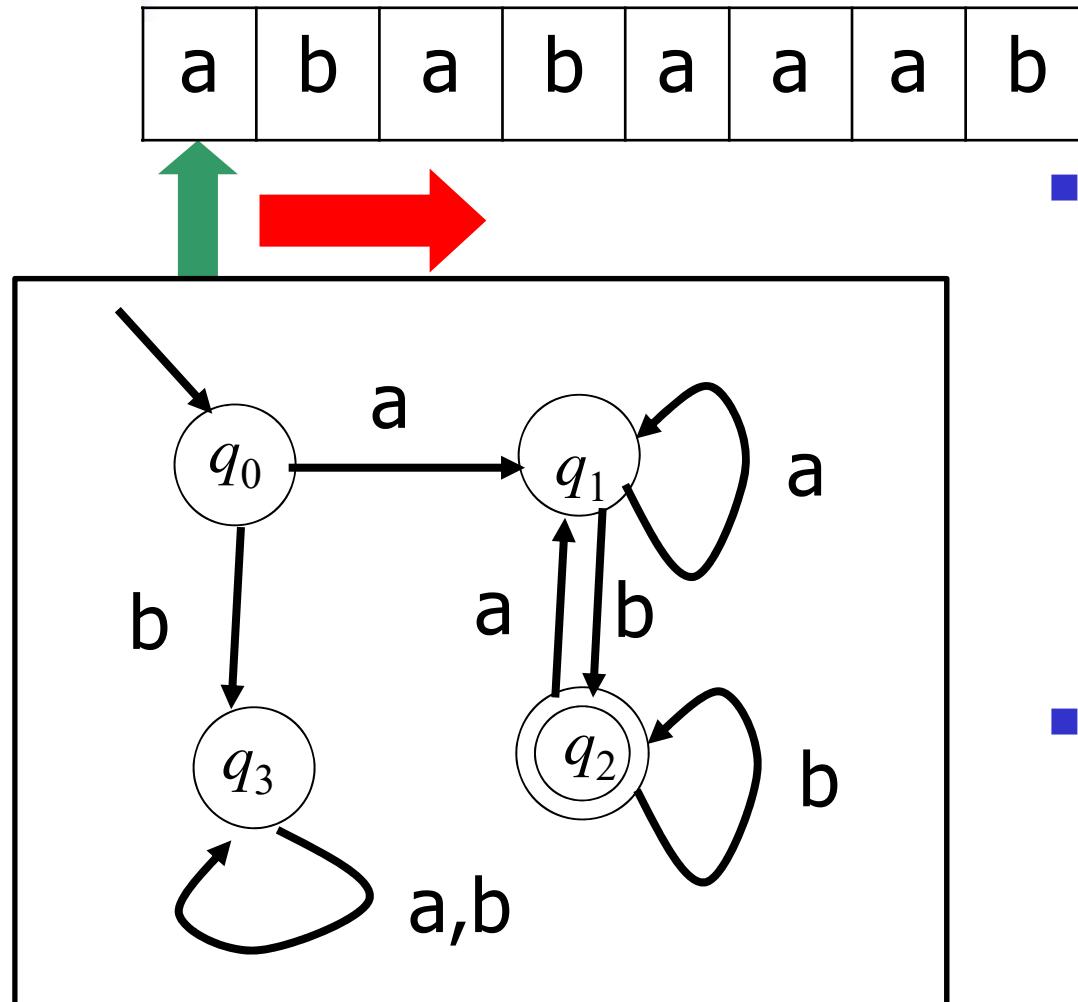
正則文法と正則言語(再)



正則文法

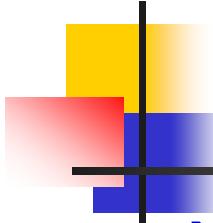
- 文法 $G = (N, \Sigma, P, S)$ は、 P 中の全ての生成規則が
 - $A \rightarrow cB$ ($A, B \in N, c \in \Sigma$) または
 - $A \rightarrow c$ ($A \in N, c \in \Sigma$) または
 - $S \rightarrow \varepsilon$ (S は開始記号)という形であるとき正則文法という

記号列の受理と言語



- テープ上の記号をすべて読み終えたときに、終了状態になっているとき、テープ上の記号列は**受理される**という。
- M によって受理される記号列全体は**言語**である。

$$L(M) = \{aab, abb, aaab, aabb, abab, \dots\}$$



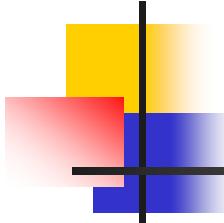
正則文法と正則言語

定理 (1) 正則文法 G が生成する言語 $L(G)$ は正則言語である。

(2) 正則言語 L を生成する正則文法 G を構成することができる。

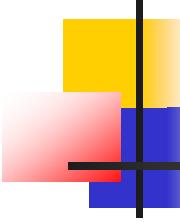


正則文法と正則言語



正則文法

- 文法 $G = (N, \Sigma, P, S)$ は、 P 中の全ての生成規則が
 - $A \rightarrow cB$ ($A, B \in N, c \in \Sigma$) または
 - $A \rightarrow c$ ($A \in N, c \in \Sigma$) または
 - $S \rightarrow \varepsilon$ (S は開始記号)という形であるとき **正則文法** という



例(2)

$G_3 = (N = \{S, A, B, C\}, \Sigma = \{a, b\},$
 $P = \{S \rightarrow aA, S \rightarrow bB, A \rightarrow aA, A \rightarrow bC, A \rightarrow b$
 $B \rightarrow aB, B \rightarrow bB, C \rightarrow aA, C \rightarrow bC, C \rightarrow b\},$
 $S)$

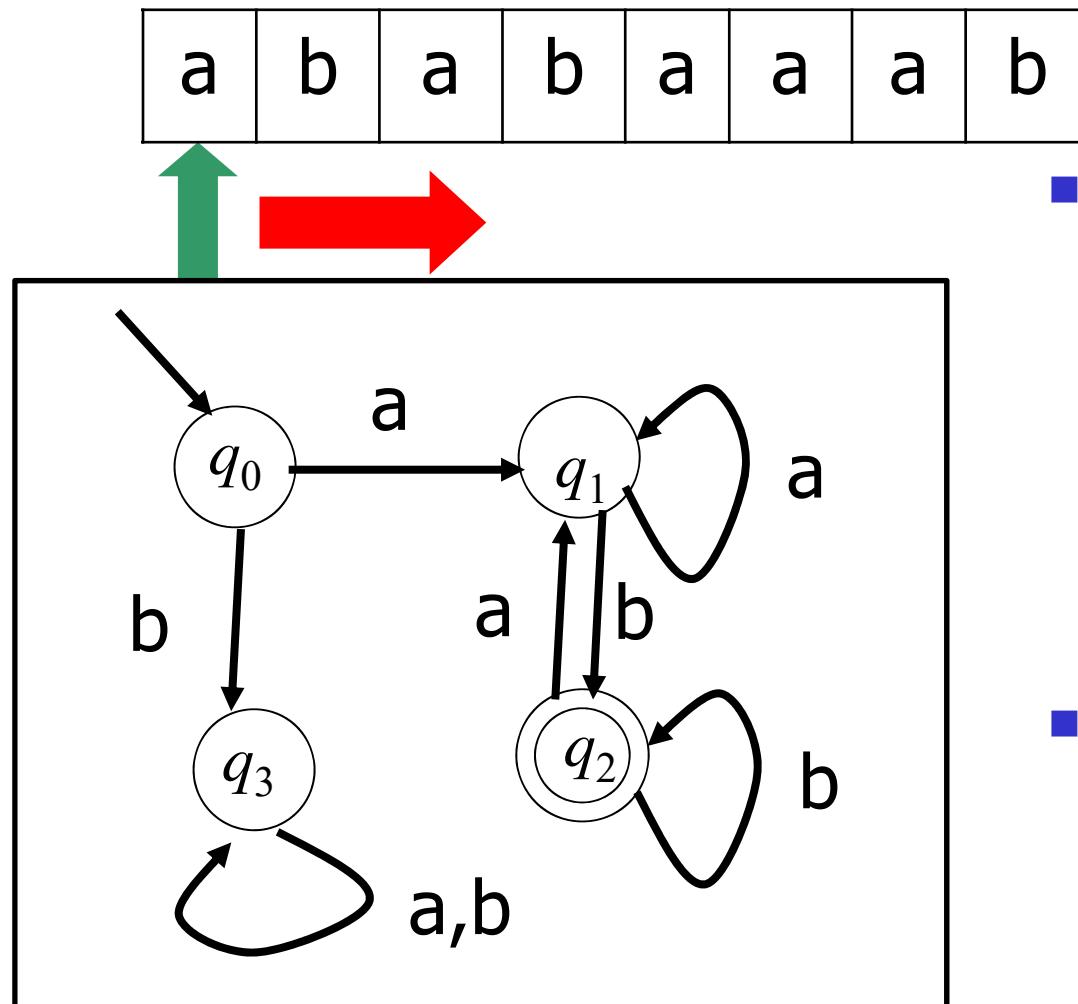
$S \Rightarrow aA \Rightarrow aaA \Rightarrow aabC \Rightarrow aab$

$S \Rightarrow aA \Rightarrow aaA \Rightarrow aabC \Rightarrow aabb$

$S \Rightarrow aA \Rightarrow abC \Rightarrow abaA \Rightarrow abab$

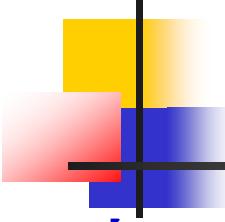
$S \Rightarrow aA \Rightarrow abC \Rightarrow abbAC \Rightarrow abbaA \Rightarrow abbaaaA$
 $\Rightarrow abbaaaA \Rightarrow abbaaaab$

記号列の受理と言語



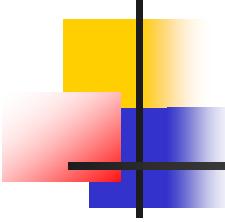
- テープ上の記号をすべて読み終えたときに、終了状態になっているとき、テープ上の記号列は**受理される**という。
- M によって受理される記号列全体は**言語**である。

$$L(M) = \{aab, abb, aaab, aabb, abab, \dots\}$$



正則文法と正則言語

- 定理** (1) 正則文法 G が生成する言語 $L(G)$ は正則言語である.
- (2) 正則言語 L を生成する正則文法 G を構成することができる.



(1)の証明

- 正則文法 $G = (N, \Sigma, P, S)$ が与えられたとき,
 $L(G) = L(M)$ を満たすような非決定性有限オートマトン
 $M = (\Sigma, Q, \delta, q_0, F)$ が構成できることを示す.

$$Q = N \cup \{q_f\}$$

$$q_0 = S$$

$$F = \{q_f\}$$

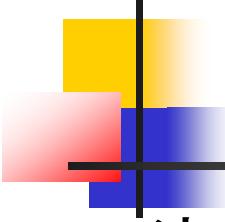
P 中に $A \rightarrow c$ という生成規則があれば

$$\delta(A, c) = \{B \mid A \rightarrow cB \in P\} \cup \{q_f\}$$

$$\delta(A, \varepsilon) = \{q_f\}$$

なければ

$$\delta(A, c) = \{B \mid A \rightarrow cB \in P\}$$



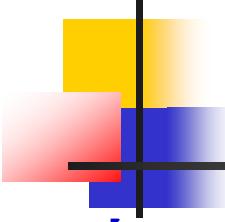
(2)の証明

- 決定性有限オートマトン $M = (\Sigma, Q, \delta, q_0, F)$ が与えられたとき, $L(G)=L(M)$ を満たすような正則文法 $G = (N, \Sigma, P, S)$ が構成できることを示す.

$$N = Q$$

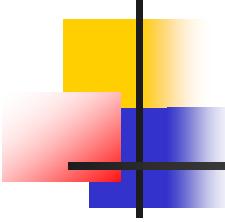
$$S = q_0$$

$$\begin{aligned}P &= \{q_i \rightarrow c q_j \mid \delta(q_i, c) = q_j\} \\&\cup \{q_i \rightarrow c \mid \delta(q_i, c) = q_j \text{かつ } q_j \in F\}\end{aligned}$$



正則文法と正則言語

- 定理** (1) 正則文法 G が生成する言語 $L(G)$ は正則言語である.
- (2) 正則言語 L を生成する正則文法 G を構成することができる.

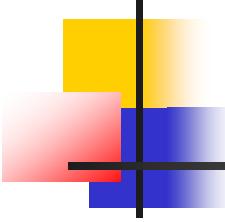


正則言語の特徴

Pumping Lemma 要素数が有限ではない正則言語 L に対して、自然数 N があって、 $s \in L$ かつ $|s| \geq N$ なるすべての語 s は $s=uvw$ かつ $|w| \geq 1$ かつすべての $i \geq 1$ に対して $uv^iw \in L$ が成り立つような部分語 w を含んでいる。

- N は L に依存して定まる(L が変わると N は変わる)
- 実は N は $L=L(M)$ なる(状態数最小の)DFA M の状態数+1とすればよい。

証明のアイデア $L=L(M)$ なる(状態数最小の)DFA $M = (\Sigma, S, d, q_0, F)$ の状態数を N' とする。 L は要素数が有限ではないので必ず L には $|s| \geq N'+1$ を満たす記号列を含む。すると M が s を受理する過程 $q_0 \Rightarrow q_1 \Rightarrow \dots \Rightarrow q_{|s|}$ を考えると鳩の巣原理から $q_i=q_j$ が成り立つ q_i, q_j ($i \neq j$)があるはず。



正則言語ではない言語

$$G_1 = (N = \{S\}, \Sigma = \{a, b\}, P = \{S \rightarrow ab, S \rightarrow aSb\}, S)$$

$$L(G_1) = \{w \in \{a, b\}^* \mid a^n b^n, n \in \mathbf{N}^+ \}$$

$L(G_1)$ は正則言語ではない。

($\Leftrightarrow L(G_1)$ を受理するDFAを構成することはできない)

証明の方針 (背理法を用いる) $L(G_1) = L(M)$ を満たす

DFA M が構成できたと仮定し、その状態数を N' とする。

このとき $n = N'/2 + 1$ とおくと $|a^n b^n| \geq N' + 1$ だから

$a^n b^n = uvw$ かつ $|w| \geq 1$ かつ

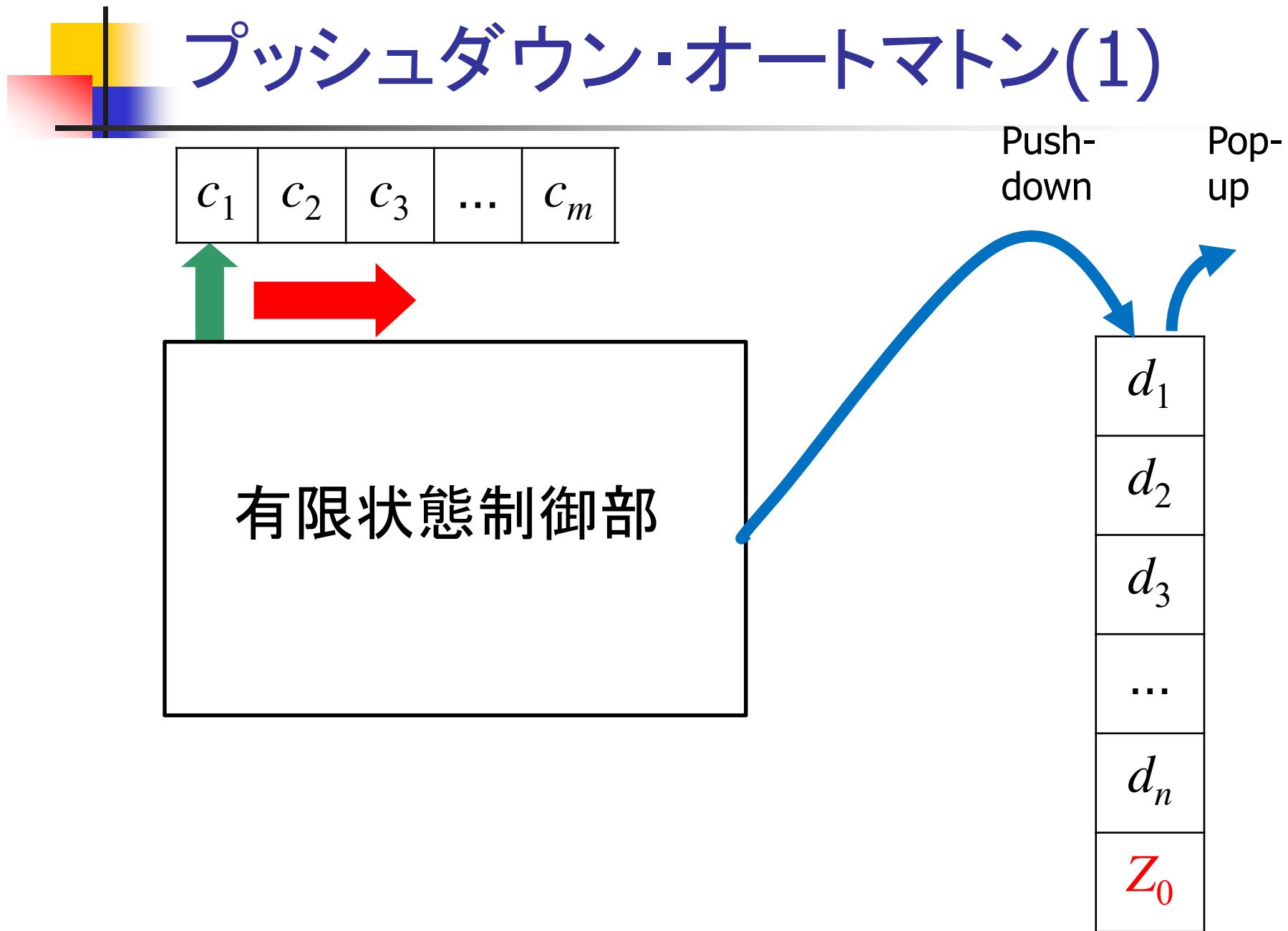
すべての $i \geq 1$ に対して $uv^i w \in L$

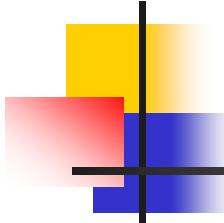
と分解できるはずである。このとき、 $uv^2 w \notin L$ であること
を $|u|$ と $|w|$ で場合分けして示す。



プッシュダウン・オートマトン

プッシュダウン・オートマトン(1)





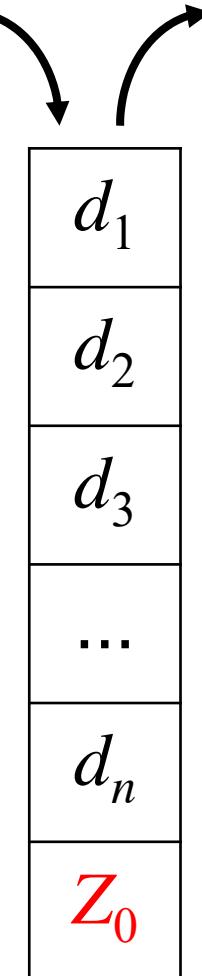
プッシュダウン・オートマトン(2)

- ヘッドが一方向(右)にしか動かない
- テープは一方向に無限長
- 入力記号列は有限長(残りは空白で埋められる)
- 記号の書換えを行わない
- 補助記憶装置としてスタックを持っている
- **2種類の受理の定義方法**
 - 入力記号列を読み終えたときに終了状態にあれば
 入力記号列は“受理”，そうでなければ“不受理”
 - 入力記号列を読み終えたときにスタックが空であれば
 入力記号列は“受理”，そうでなければ“不受理”

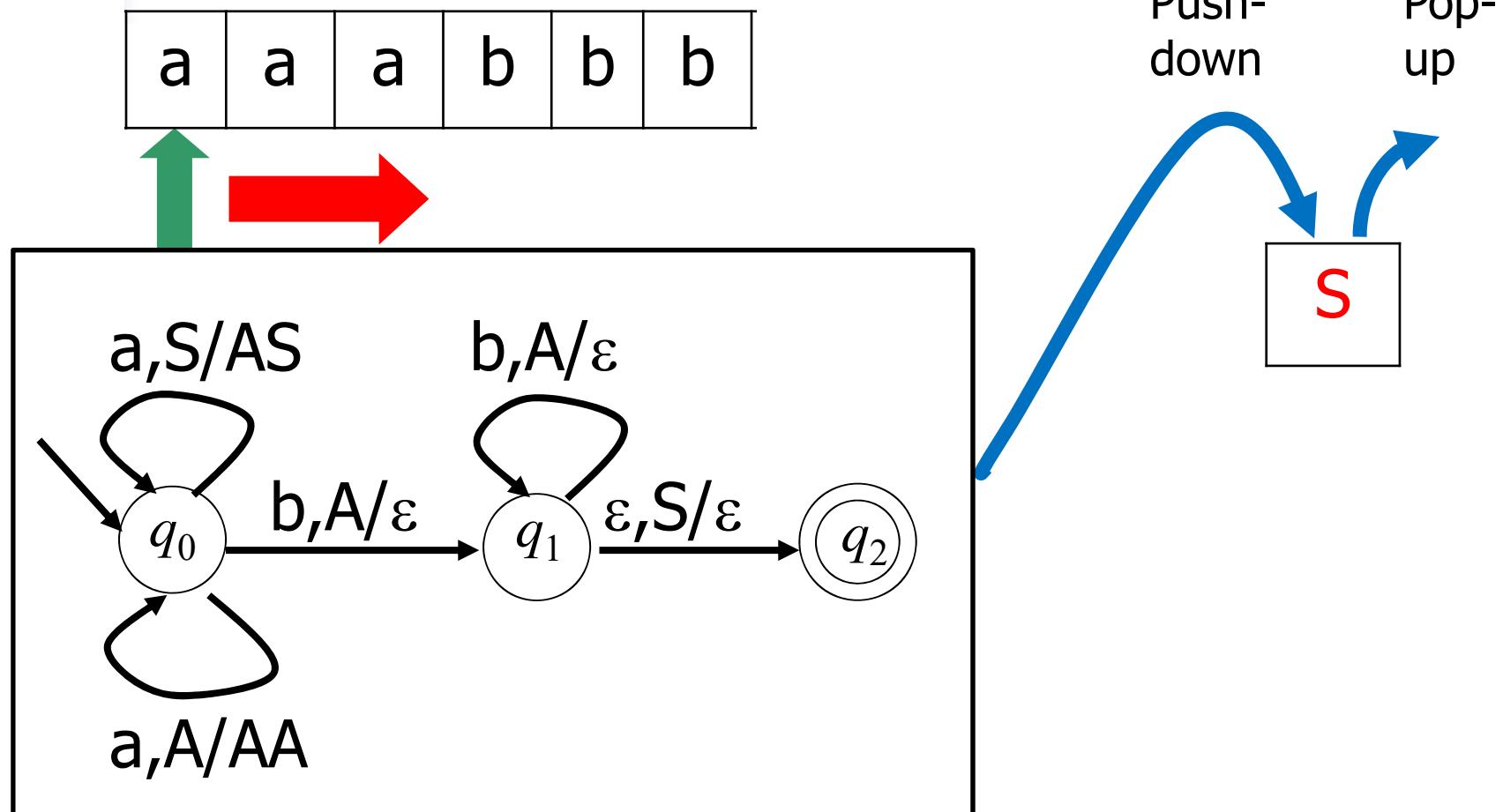
スタック(Stack)

- 有限個の記号(スタック記号)を一列に保持する
- 保持できる記号の数に上限はない
- 記号の出し入れは先頭要素だけ
 - 複数個の記号を入れるときは一つずつ入る(push-down)
 - 複数個の記号を取り出すときも一つずつ取り出す(pop-up)
- 記号を取り出すときは、最後に入れた記号を取り出す
 - 保持されている記号列の最後には“底”を示す特別な記号 Z_0 が置いてある。

Push-down Pop-up

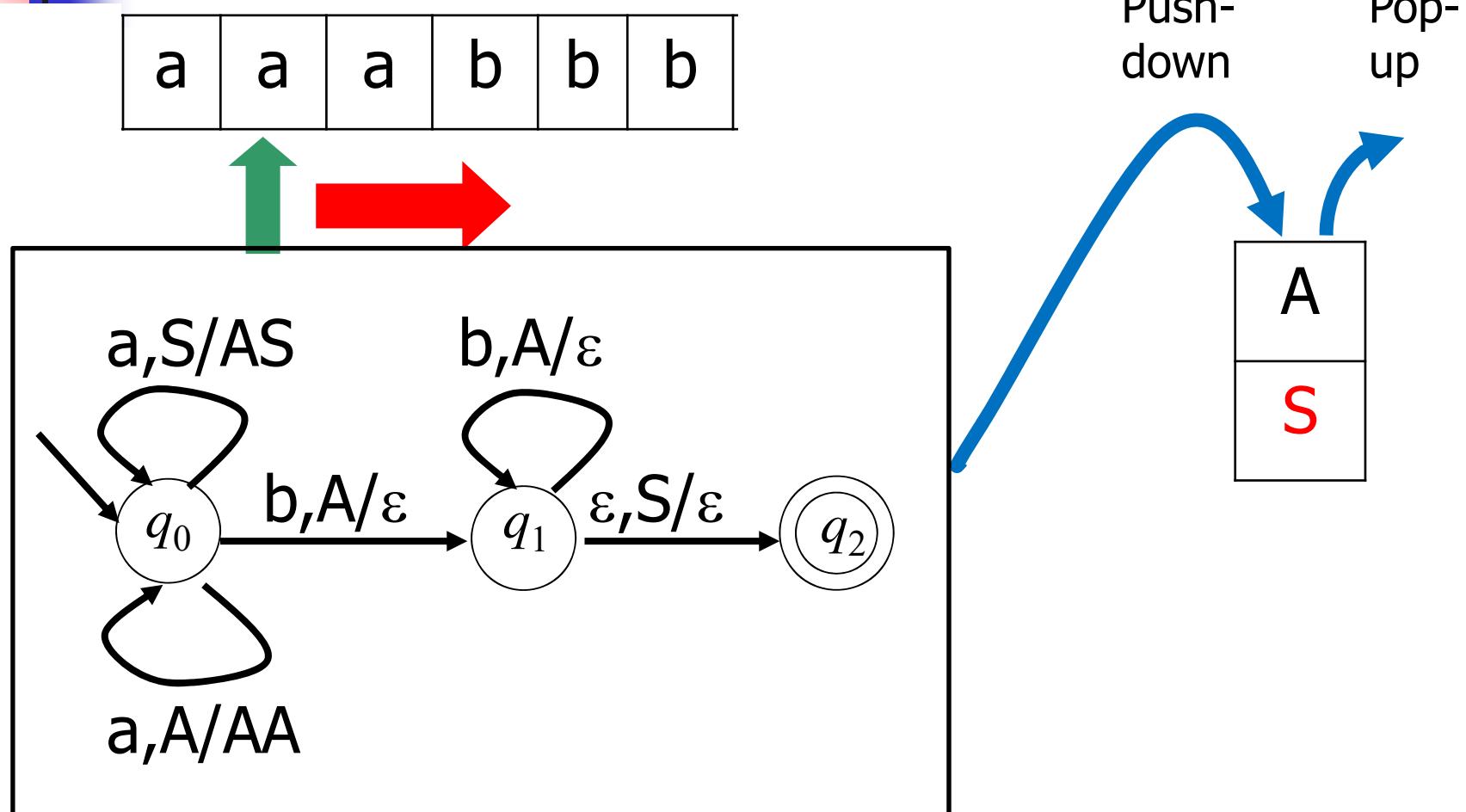


プッシュダウン・オートマトン(例1)



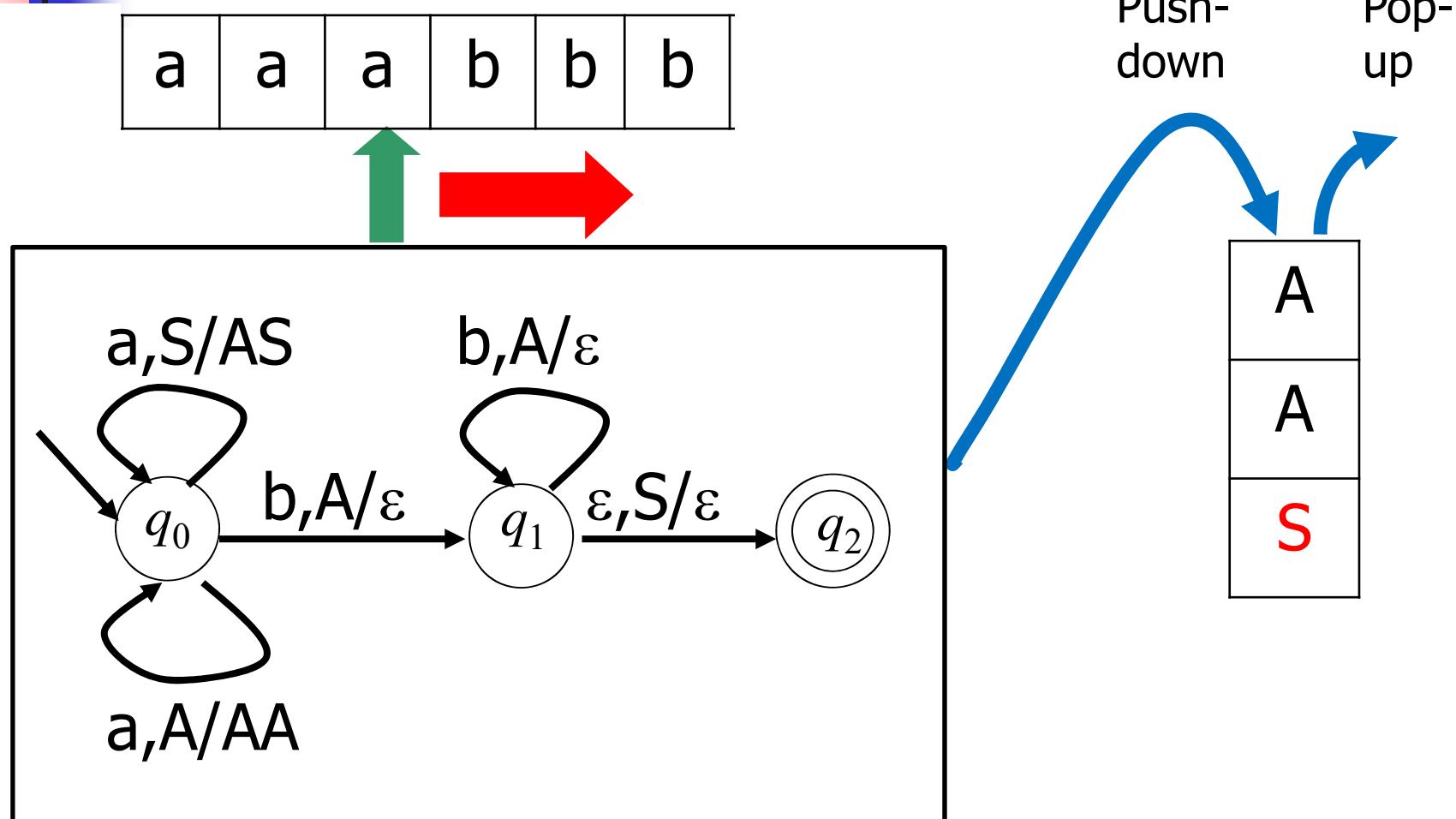
(q_0, S)

プッシュダウン・オートマトン(例2)



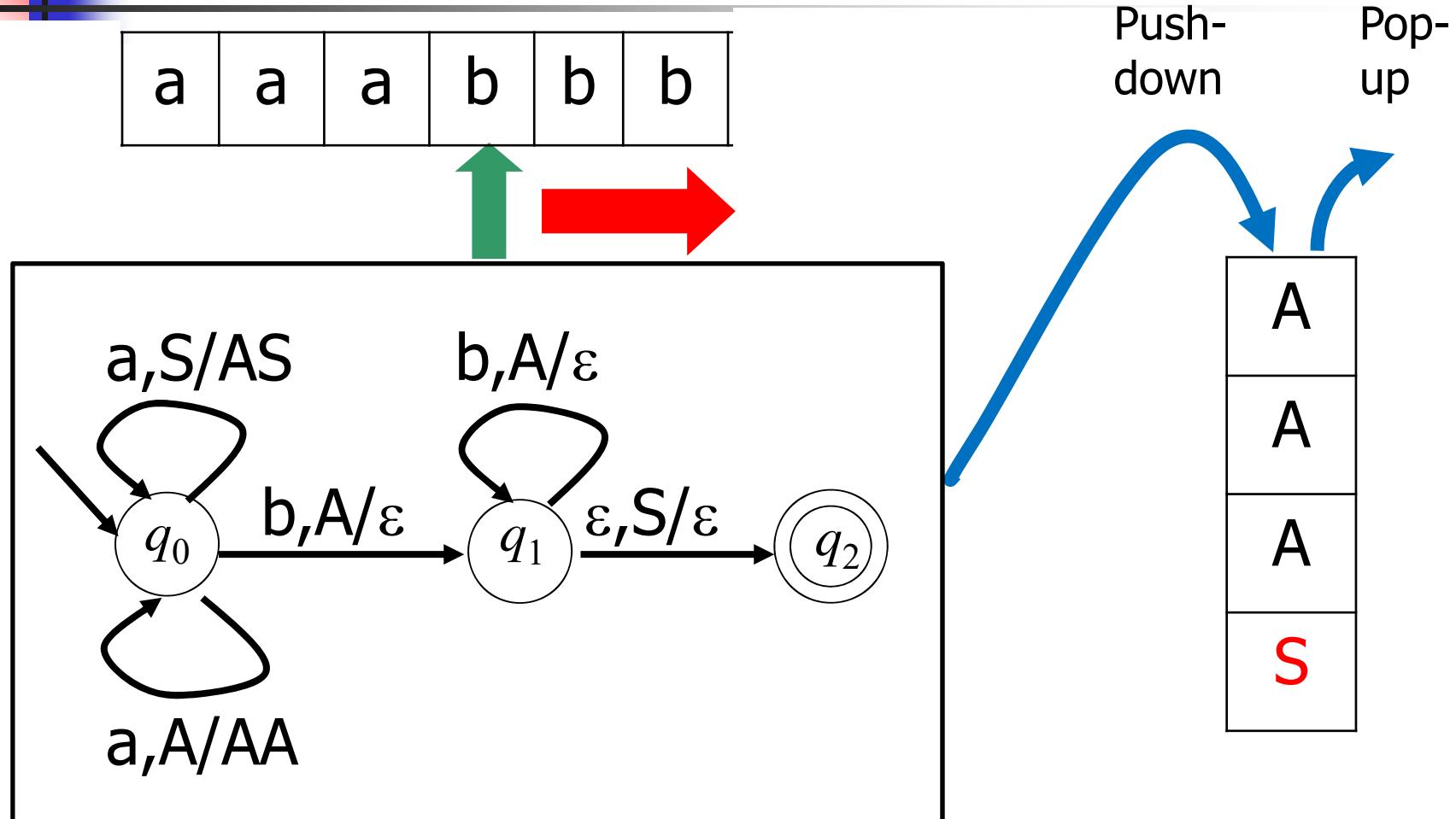
$$(q_0, S) \Rightarrow (q_0, AS)$$

プッシュダウン・オートマトン(例3)



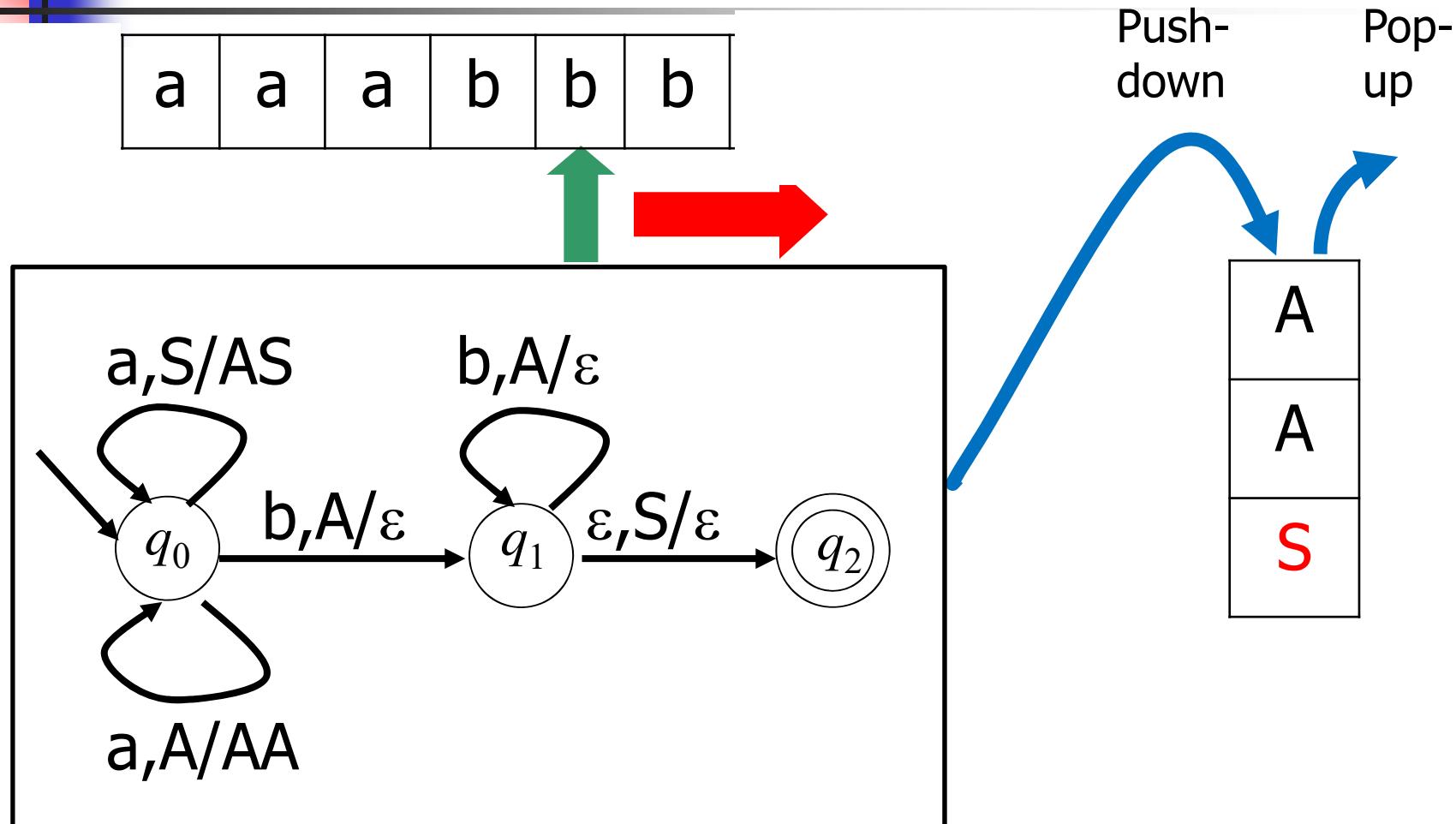
$$(q_0, S) \Rightarrow (q_0, AS) \Rightarrow (q_0, AAS)$$

プッシュダウン・オートマトン(例4)



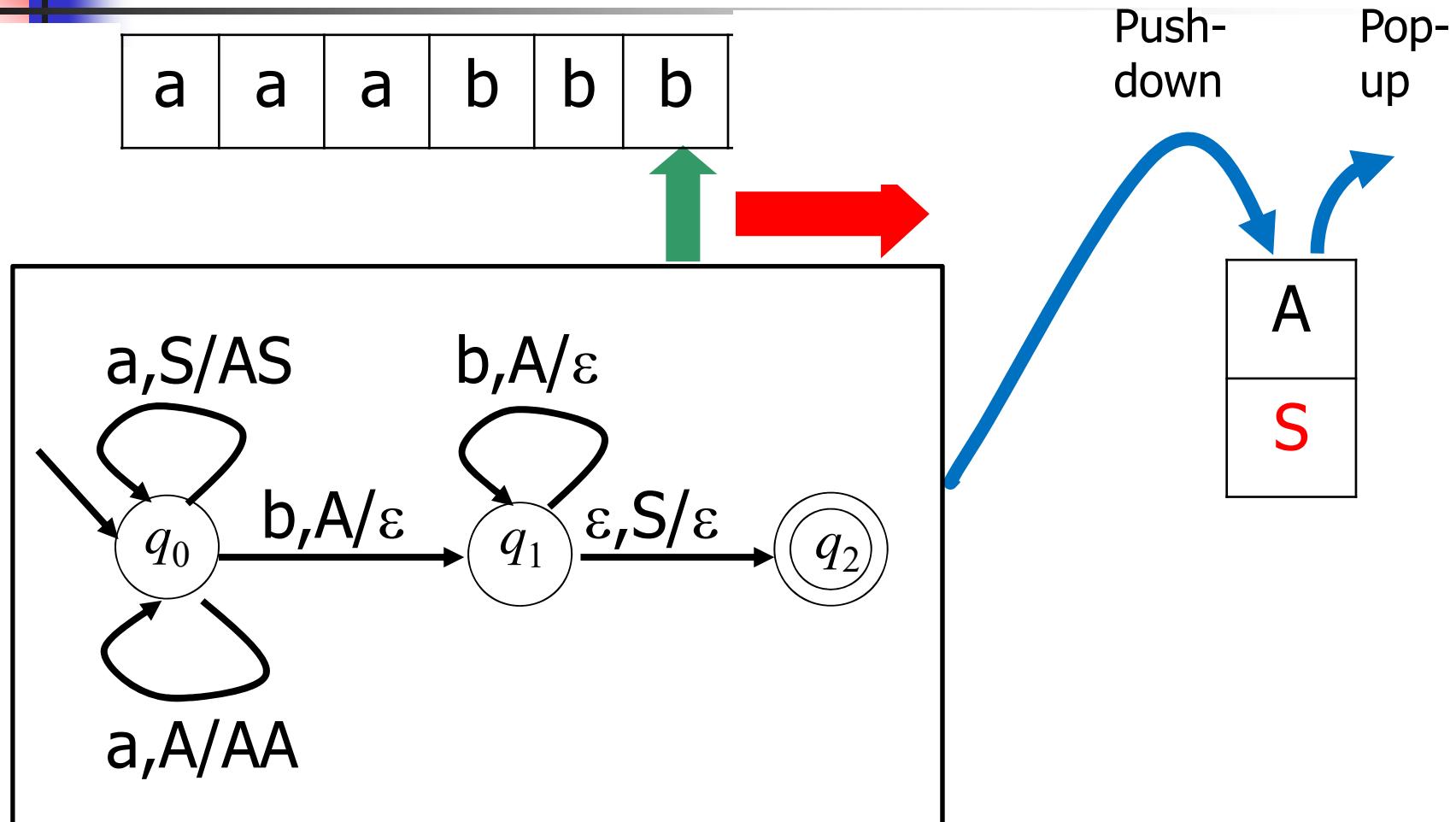
$$(q_0, S) \Rightarrow (q_0, AS) \Rightarrow (q_0, AAS) \Rightarrow (q_0, AAAS)$$

プッシュダウン・オートマトン(例5)



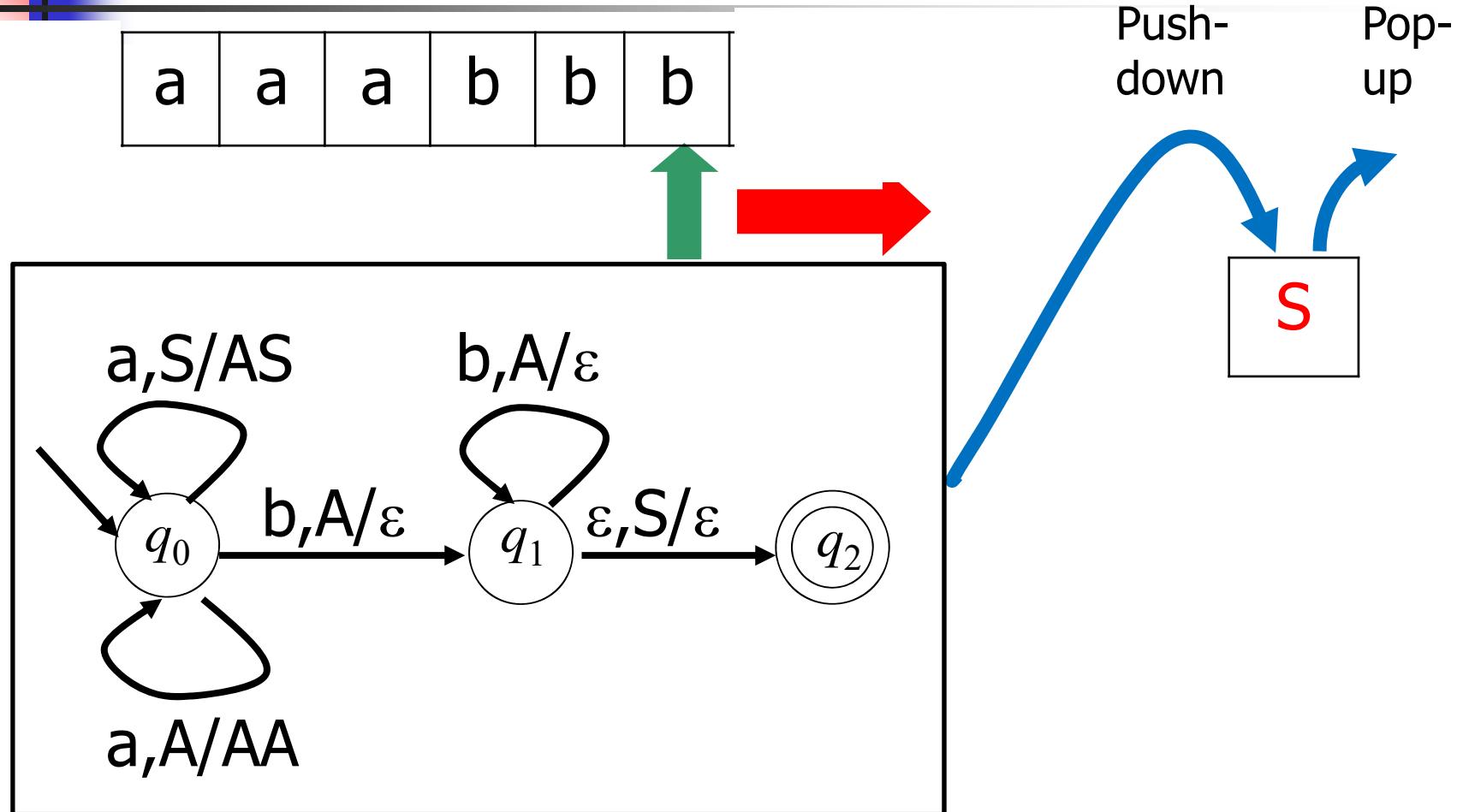
$$(q_0, S) \Rightarrow (q_0, AS) \Rightarrow (q_0, AAS) \Rightarrow (q_0, AAAS) \\ \Rightarrow (q_1, AAS)$$

プッシュダウン・オートマトン(例6)



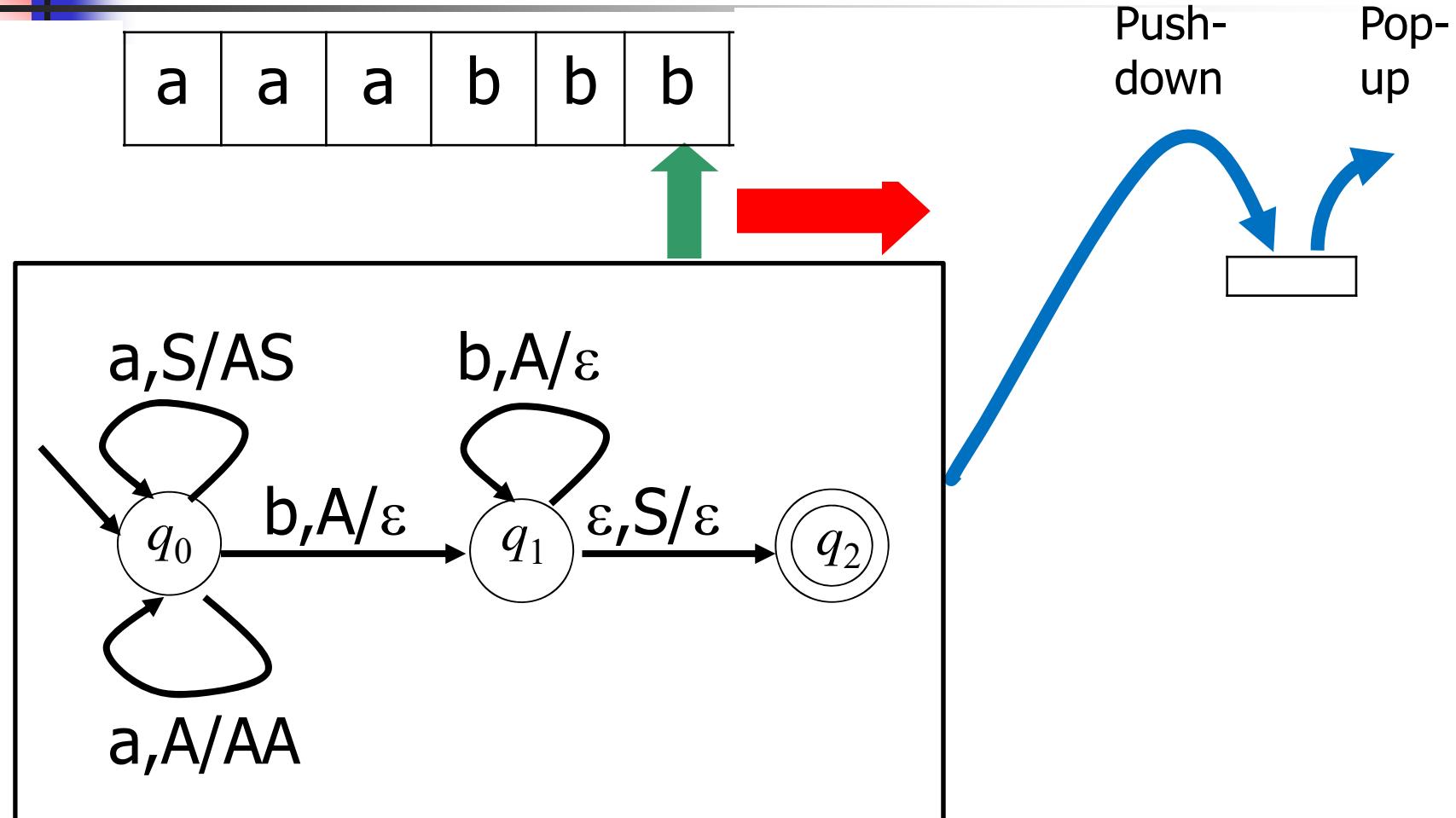
$$(q_0, S) \Rightarrow (q_0, AS) \Rightarrow (q_0, AAS) \Rightarrow (q_0, AAAS) \\ \Rightarrow (q_1, AAS) \Rightarrow (q_1, AS)$$

プッシュダウン・オートマトン(例7)

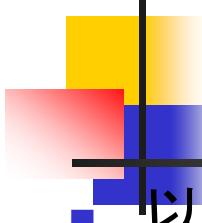


$(q_0, S) \Rightarrow (q_0, AS) \Rightarrow (q_0, AAS) \Rightarrow (q_0, AAAS)$
 $\Rightarrow (q_1, AAS) \Rightarrow (q_1, AS) \Rightarrow (q_1, S)$

プッシュダウン・オートマトン(例8)



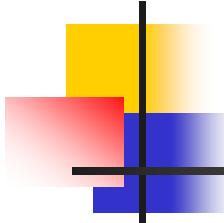
$$(q_0, S) \Rightarrow (q_0, AS) \Rightarrow (q_0, AAS) \Rightarrow (q_0, AAAS) \\ \Rightarrow (q_1, AAS) \Rightarrow (q_1, AS) \Rightarrow (q_1, S) \Rightarrow (q_2, \varepsilon)$$



集合と関数によるプッシュダウン・オートマトンの表現

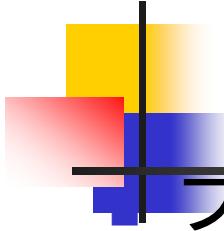
以下のような構成要素からなる組 $M=(\Sigma, \Gamma, S, \delta, s_0, Z_0, F)$ を
決定性 プッシュダウン・オートマトンとよぶ.

- S は空でない有限集合であり,
その要素を状態とよぶ.
- Σ は入力アルファベット
- Γ はスタック・アルファベット
 - Γ 中の記号をスタック記号とよぶ
- δ は $S \times \Sigma \cup \{\varepsilon\} \times \Gamma \rightarrow S \times \Gamma^*$ なる**部分** 関数
- $s_0 \in S$ は特定の状態であり, 初期状態とよぶ.
- $Z_0 \in \Gamma$ は特定のスタック記号であり, 底記号と呼ぶ
- $F \subset S$ は特定の状態の集合であり, その要素を終了状態,
もしくは受理状態とよぶ.



注意

- 決定性ではあるが、
 - 遷移関数 δ が定義されないような組 $S \times \Sigma \cup \{\varepsilon\} \times \Gamma$ を認める。(δ は部分関数)
 - ε 遷移を認める
- $d \in \Gamma$ に対して, $\delta(s, \varepsilon, d)$ が定義されているとき,
- $c \in \Sigma$ に対しては $\delta(s, c, d)$ は定義しない



計算状況と動作, 受理

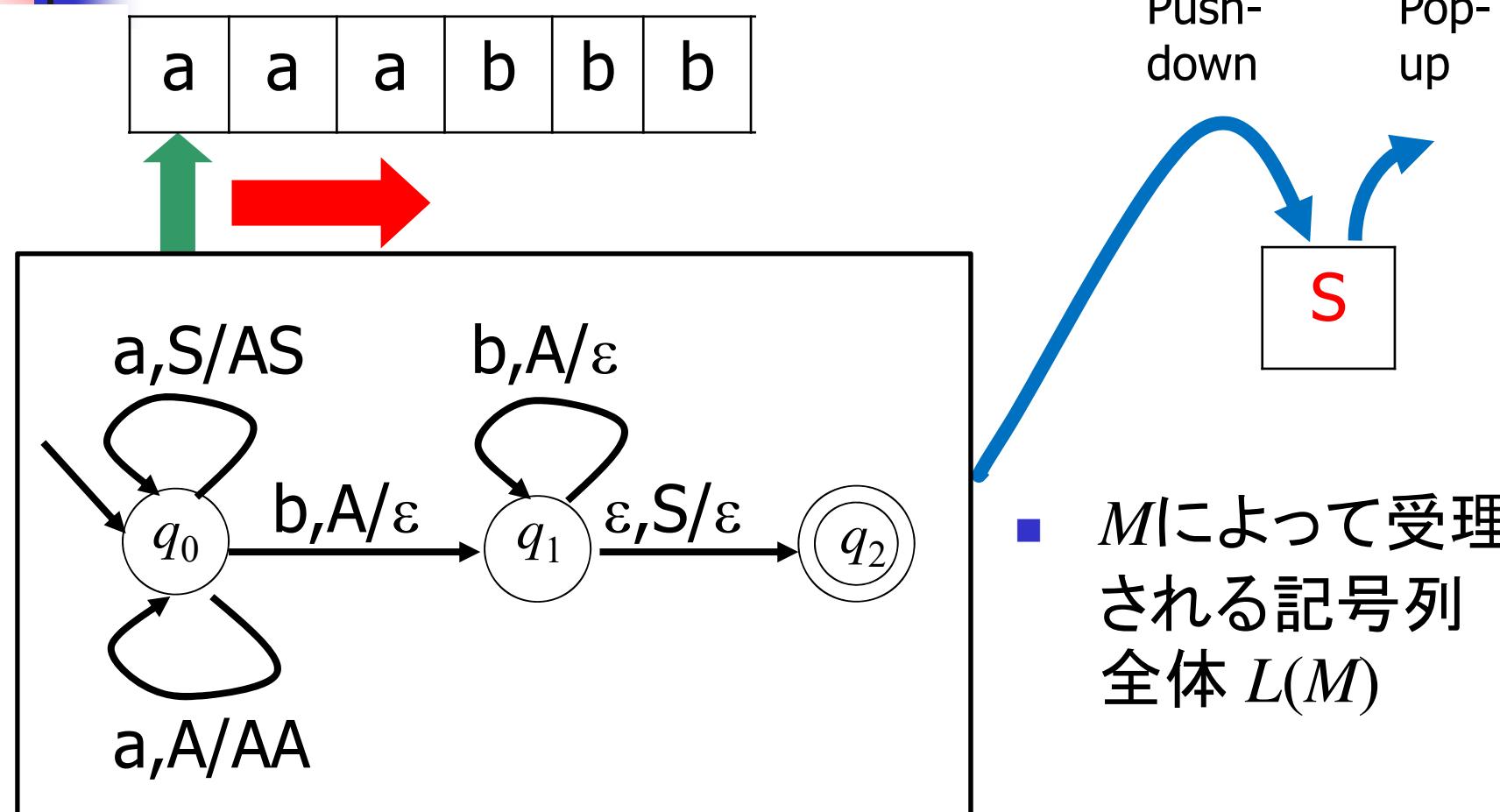
プッシュダウン・オートマトンの計算状況は

状態 $s \in \Sigma$ とスタックに保持されているスタック記号
 $\gamma \in \Gamma^*$ の組 (s, γ) で表す.

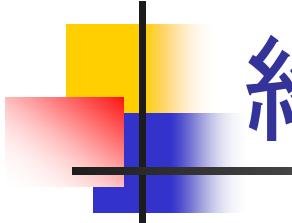
- スタックの先頭要素を $\text{to}(\gamma)$ と表す
- 入力記号 c を読み込んだときの計算状況 (s, γ) からの動作は, $\gamma \neq \varepsilon$ かつ $\text{top}(\gamma) = d$, $\gamma = d\gamma'$ のとき,
 $\delta(s, c, d) = (t, \eta)$ ならば, $(s, \gamma) \Rightarrow (t, \eta \gamma')$ とする
 - $\gamma = \varepsilon$ の場合は動作は定義しない
- $(q_0, Z_0) \xrightarrow{c_1} \dots \xrightarrow{c_n} (q_f, \varepsilon)$ かつ $q_f \in F$ のとき, M は $c_1 \dots c_n$ を受理する

例 $(q_0, S) \Rightarrow (q_0, AS) \Rightarrow (q_0, AAS) \Rightarrow (q_0, AAAS)$
 $\Rightarrow (q_1, AAS) \Rightarrow (q_1, AS) \Rightarrow (q_1, S) \Rightarrow (q_2, \varepsilon)$

PDAが受理する言語



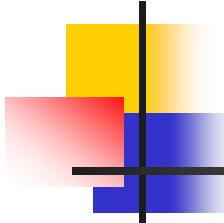
$$L(M) = \{ab, aabb, aaabbb, aaaabbbb, \dots\}$$



終了状態による受理

- 入力記号列を読み終えたときに、スタックが空であるか否かにかかわらず
 - 終了状態にあれば、入力記号列は“受理”
 - 終了状態になければ、入力記号列は“不受理”

$(q_0, Z_0) \xrightarrow{c_1} \dots \xrightarrow{c_n} (q_f, \gamma_n)$ かつ $q_f \in F$ のとき、 M は $c_1 \dots c_n$ を
受理

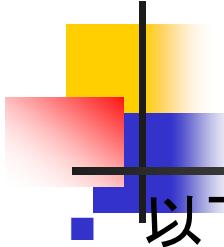


空スタックによる受理

- 決定性プッシュダウンオートマトンの定義に終了状態の集合を使わない: $M = (\Sigma, \Gamma, S, \delta, s_0, Z_0)$
- 入力記号列を読み終えたときに,
 - スタックが空になつていれば、入力記号列は“受理”
 - スタックが空になつていなければ入力記号列は“不受理”

$(q_0, Z_0) \xrightarrow{c_1} \dots \xrightarrow{c_n} (q_f, \varepsilon)$ のとき, M は $c_1 \dots c_n$ を受理

注意 空スタックによる記号列を受理するDPDA M を終了状態によって受理するDPDA M' に変換することは可能である。その逆は一般には不可能である。

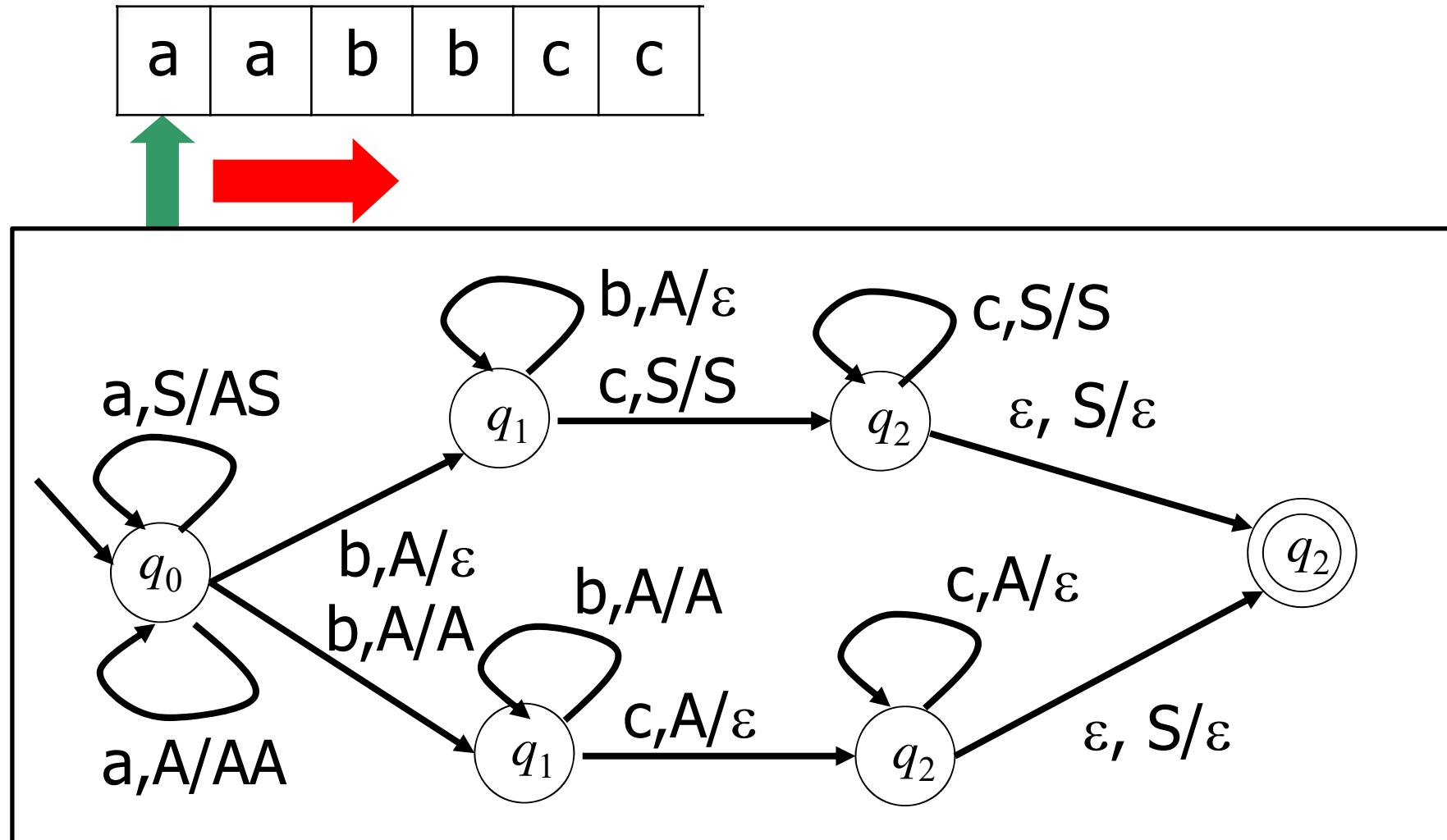


非決定性プッシュダウン・オートマトン

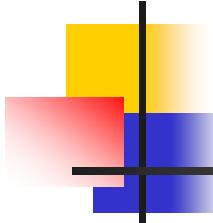
以下のような構成要素からなる組 $M=(\Sigma, \Gamma, S, \delta, s_0, Z_0, F)$ を
非決定性プッシュダウン・オートマトンとよぶ.

- S は空でない有限集合であり,
その要素を状態とよぶ.
- Σ は入力アルファベット
- Γ はスタック・アルファベット
 - Γ 中の記号をスタック記号とよぶ
- δ は $S \times \Sigma \cup \{\varepsilon\} \times \Gamma \rightarrow 2^{S \times \Gamma^*}$ なる関数
- $s_0 \in S$ は特定の状態であり, 初期状態とよぶ.
- $Z_0 \in \Gamma$ は特定のスタック記号であり, 底記号と呼ぶ
- $F \subset S$ は特定の状態の集合であり, その要素を終了状態,
もしくは受理状態とよぶ.

非決定性PDA(例)



$L(M) = \{a^n b^m c^k \mid n, m, k \in \mathbb{N}^+ \text{かつ } n=m \text{ または } n=k\}$



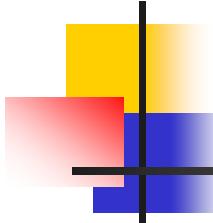
注意(1)

- 決定性ではあるが、
 - 遷移関数 δ を部分関数にする必要はない.
 - ε 遷移を認める.

$d \in \Gamma$ に対して, $\delta(s, \varepsilon, d)$ が定義されているとき

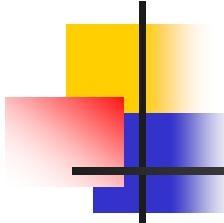
,

$c \in \Sigma$ に対しては $\delta(s, c, d)$ を定義してよい



注意(2)

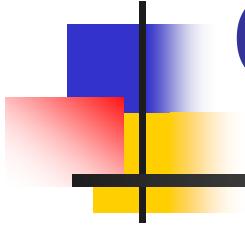
- 非決定性プッシュダウン・オートマトン M が終了状態により記号列を受理すると定義したとき, 空スタックにより記号列を受理すると定義して同じ言語を受理する非決定性プッシュダウン・オートマトン M' を構成できる
- 非決定性プッシュダウン・オートマトン M が空スタックにより記号列を受理すると定義したとき, 終了状態により記号列を受理すると定義して同じ言語を受理する非決定性プッシュダウン・オートマトン M' を構成できる



DPDAとNPDA

定理 NPDA M が受理する言語 $L(M)$ で, どのような DPDA も $L(M)$ を受理することができないものが存在する.

例 $L(M) = \{w(w^R)^k \mid w \in \{a, b\}^*\}$ を受理する DPDA は構成できない.



Greibach標準形のCFGとNPDA



Greibach標準形の文法

- 文脈自由文法 $G = (N, \Sigma, P, S)$ は、 P 中の全ての生成規則が次の 3 種類のいずれかであるとき、Greibach 標準形であるという。

$$A \rightarrow cB_1 \dots B_n \quad A \rightarrow c \quad S \rightarrow \varepsilon$$

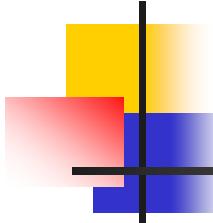
ただし $B_1, \dots, B_n \in N$, $c \in \Sigma$

また $S \rightarrow \varepsilon \in P$ ならば $B_1, \dots, B_n \in N - \{S\}$

例 $G_1'' = (\{S, T, A, B\}, \{a, b\}, P, S)$

$$\begin{aligned} P = & \{S \rightarrow aT, S \rightarrow aB, T \rightarrow aTB, B \rightarrow aBB, \\ & A \rightarrow a, B \rightarrow b\} \end{aligned}$$

$$L(G_1'') = \{a^n b^n \mid n \in \mathbf{N}^+\}$$

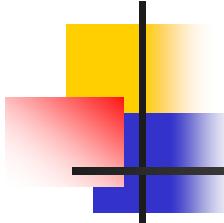


Greibach標準形への変換

1. $G = (N, \Sigma, P, S)$ をChomsky標準形に変換する
 - 非終端記号に全順序を仮定する
 - 説明を簡単にするため, A_i の添え字が順序を表していると仮定する $A_1 < A_2 < \dots < A_n$

非終端記号は

2. G の左再帰性を除去する(上り)
3. G の生成規則をGreibach標準形に整える(下り)



生成規則の置換

補題 文脈自由文法 $G = (N, \Sigma, P, S)$ の P 中の生成規則に

$$A \rightarrow B\gamma \quad (B \in N, \gamma \in (N \cup \Sigma)^*)$$

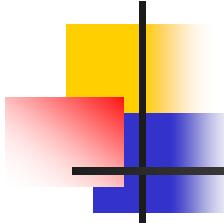
が含まれていて、左辺が B である規則が

$$B \rightarrow \delta_1, \dots, B \rightarrow \delta_n$$

であるとき、 $G' = (N, \Sigma, P', S)$

$$P' = P - \{A \rightarrow B\gamma\} \cup \{A \rightarrow \delta_1\gamma, \dots, A \rightarrow \delta_n\gamma\}$$

とすれば $L(G') = L(G)$ である



直接左再帰性の除去

補題 文脈自由文法 $G = (N, \Sigma, P, S)$ の P 中の生成規則で左辺が $A \in N$ であるものを右辺の左端が A であるものとそうでないものの 2 種類に分類しておく。

$$A \rightarrow A\gamma_1, \dots, A \rightarrow A\gamma_k \quad (\gamma_1, \dots, \gamma_k \in (N \cup \Sigma)^*)$$

$$A \rightarrow \delta_1, \dots, A \rightarrow \delta_n$$

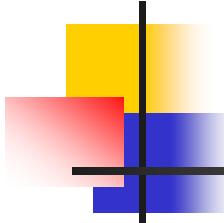
$(\delta_1, \dots, \delta_n \in (N \cup \Sigma)^*, \delta_i$ の左端は A ではない)

このとき、新しい非終端記号 Z を用いて

$$A \rightarrow \delta_1 Z, \dots, A \rightarrow \delta_n Z$$

$$Z \rightarrow \gamma_1, \dots, Z \rightarrow \gamma_k, Z \rightarrow \gamma_1 Z, \dots, Z \rightarrow \gamma_k Z$$

として、上の左辺が $A \in N$ である規則を置き換えて得られる文法 $G' = (N \cup \{Z\}, \Sigma, P, S)$ は、 $L(G') = L(G)$ を満たす。



左再帰生成規則の除去 (1)

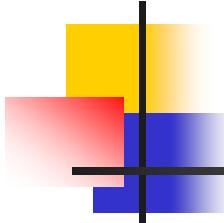
- 文脈自由文法 $G = (N, \Sigma, P, S)$

補題 P は、その中の

(*) $A_i \rightarrow A_j \gamma$ という形の生成規則はすべて $i < j$ を満たすように変形することができる。

証明 A_1 については $i = 1$ とすると、(*) が満たされないのは $j = 1$ のときだけである。このときは直接左再帰性の除去を行えばよい。

- 以下、新たに導入する非終端記号 Z_i の順序は $A_{i-1} < Z_i < A_i$ としておく



左再帰生成規則の除去 (2)

証明の続き もし $i=1, \dots, k$ について, A_i に関する規則が
(*)を満たしていたとし, $i=k+1$ について

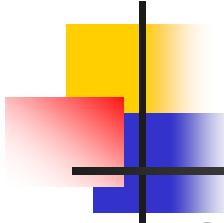
$A_{k+1} \rightarrow A_j \gamma \quad k+1 \geq j$ なる規則があったとする.

このとき, $k+1 > j$ なら生成規則の置換によって右辺の
 A_j を消去する. すると帰納法の仮定から, 右辺の非
終端記号は $A_{j+1}, A_{j+2}, \dots, A_n$ のいずれかである.

したがって, この操作を高々 k 回繰り返せば,

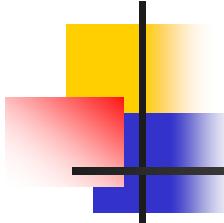
$A_{k+1} \rightarrow A_j \gamma \quad k+1 \leq j$

を満たすようにすることができる. さらに $j = k+1$ なら
ば, 直接左再帰除去を適用すればよい.



Greibach標準形への整形

- Chomsky標準形から変形したとすると $S \rightarrow \varepsilon$ 以外の生成規則は以下の3種類のいずれかにになっている
 - (1) $A_i \rightarrow A_j \gamma$ ($i < j$, $\gamma \in (N \cup \{Z_1, \dots, Z_n\})^+$)
 - (2) $A_i \rightarrow c \delta$ ($c \in \Sigma$, $\delta \in (N \cup \{Z_1, \dots, Z_n\})^*$)
 - (3) $Z_i \rightarrow \beta$ ($\beta \in (N \cup \{Z_1, \dots, Z_n\})^+$)
- 順序が最大の A_n について(1)はありえない.
- A_{n-1} について(1)の形の規則は $j = n$ であるから, 生成規則の置換によって A_n を除去すると(2)になる.
- 以下, A_{n-2}, A_{n-3}, \dots と同様の操作を行う
- (1)の形の規則は, その生成方法から β の左端の記号は A_n, A_{n-1}, \dots, A_1 のいずれかだからそれを除去する



最左導出と最右導出

- CFG $G = (N, \Sigma, P, S)$

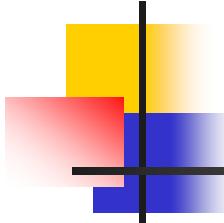
における導出

$$\alpha = \gamma_1 \Rightarrow \gamma_2 \Rightarrow \dots \Rightarrow \gamma_n = \beta$$

において、

$\gamma_1, \gamma_2, \dots, \gamma_n$ の最も左にある非終端記号が導出の対象であるとき、最左導出という

$\gamma_1, \gamma_2, \dots, \gamma_n$ の最も右にある非終端記号が導出の対象であるとき、最右導出という



Greibach標準形の文法

- 文脈自由文法 $G = (N, \Sigma, P, S)$ は、 P 中の全ての生成規則が次の3種類のいずれかであるとき、Greibach標準形であるという。

$$A \rightarrow cB_1 \dots B_n \quad A \rightarrow c \quad S \rightarrow \varepsilon$$

ただし $B_1, \dots, B_n \in N$, $c \in \Sigma$

また $S \rightarrow \varepsilon \in P$ ならば $B_1, \dots, B_n \in N - \{S\}$

例 $G_1'' = (\{S, T, A, B\}, \{a, b\}, P, S)$

$$\begin{aligned} P = \{ & S \rightarrow aT, S \rightarrow aB, T \rightarrow aTB, T \rightarrow aTB, T \rightarrow aBB, \\ & A \rightarrow a, B \rightarrow b \} \end{aligned}$$

$$L(G_1'') = \{a^n b^n \mid n \in \mathbf{N}^+\}$$

Greibach標準形による構文解析

$$P = \{S \rightarrow aT, S \rightarrow aB, T \rightarrow aTB, T \rightarrow aBB, A \rightarrow a, B \rightarrow b\}$$

aaaabbbb

$S \rightarrow aB \rightarrow ab$

$aT \rightarrow aaBB \rightarrow aabB \rightarrow aabb$

$aaTB \rightarrow aaaBBB \rightarrow aaabBB \rightarrow aaabbB$

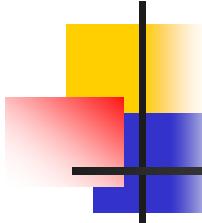
$\rightarrow aaabbb$

$aaaTBB \rightarrow aaaaBBBB \rightarrow aaaabBBB$

$\rightarrow aaaabbBB$

$\rightarrow aaaabbbB$

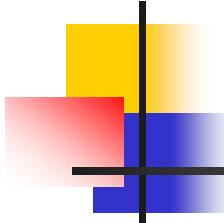
$\rightarrow aaaabbbb$



Greibach標準形からNPDAへの変換

定理 Greibach標準形の文脈自由文法 $G = (N, \Sigma, P, S)$ に対して, NPDA $M = \{\{q\}, N, \Sigma, \delta, q, S, \emptyset\}$ は $L(G) = L(M)$. ただし,

$$\delta(q, c, A) = \{(q, B_1 \dots B_n) \mid A \rightarrow cB_1 \dots B_n \in P\}$$



NPDAからGreibach標準形への変換

定理 空スタックで受理するNPDA $M = \{Q, \Gamma, \Sigma, \delta, q_0, Z_0, \emptyset\}$ に対して、以下のように Greibach標準形の文脈自由文法 $G = (N, \Sigma, P, S)$ を定義すれば $L(G) = L(M)$.

$$N = \{S\} \cup \{[q, c, r] \mid q, r \in Q, c \in \Sigma\}$$

$$P = \{ S \rightarrow [q_0, Z_0, r] \mid r \in Q\}$$

$$\begin{aligned} & \cup \left(\bigcup_{q \in Q, c \in \Sigma, A \in \Gamma} \{ [q, A, r] \rightarrow c[r_1, B_1, r_2][r_2, B_2, r_3] \dots [r_n, B_n, r] \right. \\ & \quad \left. \mid cB_1 \dots B_n \in \delta(q, c, A), r, r_1, \dots, r_n \in Q \} \right) \end{aligned}$$

導出木の例(2)

$N = \{ \text{式} \}$

$\Sigma = \{ a, b, c, +, *, (,) \}$

開始記号 : 式

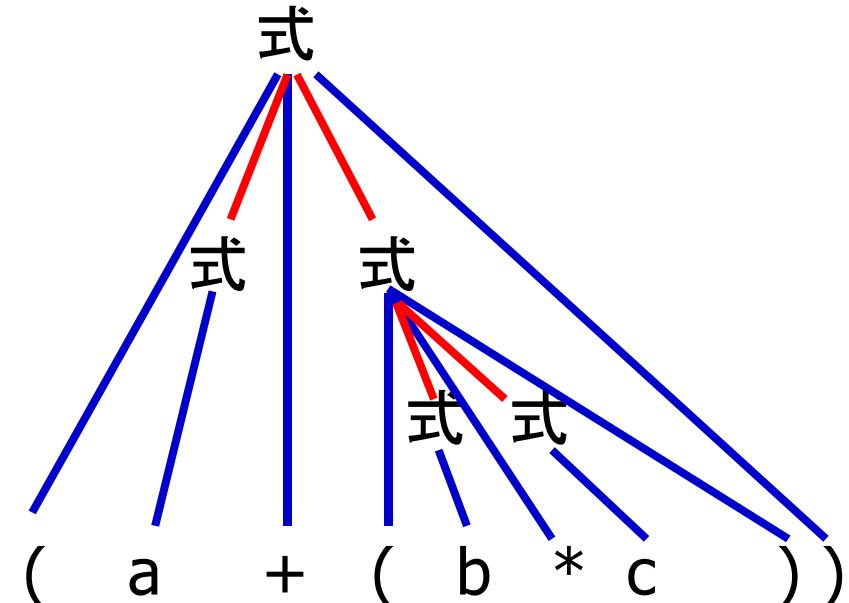
$P = \{ \text{式} \rightarrow (\text{式} + \text{式}),$

$\text{式} \rightarrow (\text{式} * \text{式}),$

$\text{式} \rightarrow a,$

$\text{式} \rightarrow b,$

$\text{式} \rightarrow c \}$

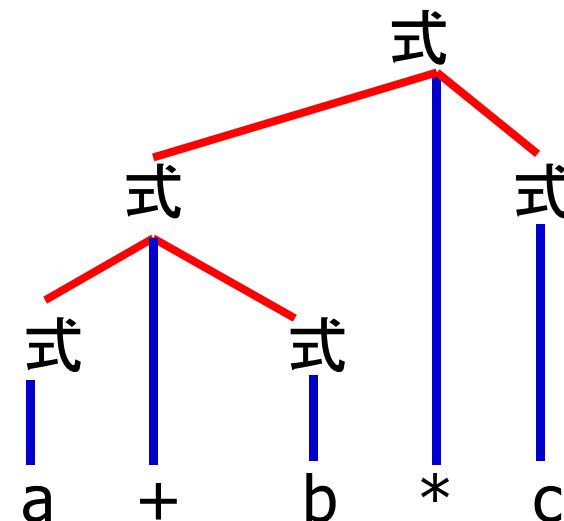
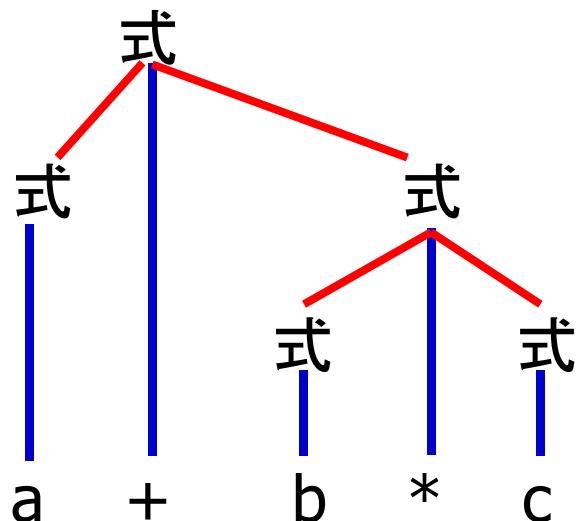


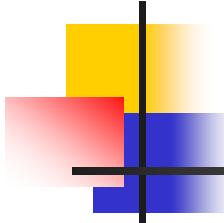
$\text{式} \Rightarrow (\text{式} + \text{式}) \Rightarrow (a + \text{式}) \Rightarrow (a + (\text{式} * \text{式}))$

$\Rightarrow (a + (b * \text{式})) \Rightarrow (a + (b * c))$

曖昧な文法

- ある文法を用いると、一つの単語(記号)の列 σ に対して異なる2種類以上の導出が構成されることがあるとき、その文法は**曖昧**であるという。





最左導出と最右導出

- CFG $G = (N, \Sigma, P, S)$

における導出

$$\alpha = \gamma_1 \Rightarrow \gamma_2 \Rightarrow \dots \Rightarrow \gamma_n = \beta$$

において、

$\gamma_1, \gamma_2, \dots, \gamma_n$ の最も左にある非終端記号が導出の対象であるとき、最左導出という

$\gamma_1, \gamma_2, \dots, \gamma_n$ の最も右にある非終端記号が導出の対象であるとき、最右導出という



Greibach標準形の文法

- 文脈自由文法 $G = (N, \Sigma, P, S)$ は、 P 中の全ての生成規則が次の 3 種類のいずれかであるとき、Greibach 標準形であるという。

$$A \rightarrow cB_1 \dots B_n \quad A \rightarrow c \quad S \rightarrow \varepsilon$$

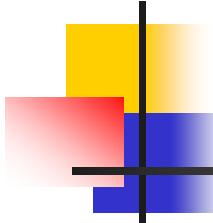
ただし $B_1, \dots, B_n \in N$, $c \in \Sigma$

また $S \rightarrow \varepsilon \in P$ ならば $B_1, \dots, B_n \in N - \{S\}$

例 $G_1'' = (\{S, T, A, B\}, \{a, b\}, P, S)$

$$\begin{aligned} P = \{ & S \rightarrow aT, S \rightarrow aB, T \rightarrow aTB, T \rightarrow aTB, T \rightarrow aBB, \\ & A \rightarrow a, B \rightarrow b \} \end{aligned}$$

$$L(G_1'') = \{a^n b^n \mid n \in \mathbf{N}^+\}$$



Greibach標準形による構文解析

$$P = \{S \rightarrow aT, S \rightarrow aB, T \rightarrow aTB, T \rightarrow aBB, A \rightarrow a, B \rightarrow b\}$$

aaaabbbb

$S \rightarrow aB \rightarrow ab$

$\searrow aT \rightarrow aaBB \rightarrow aabB \rightarrow aabb$

$aaTB \rightarrow aaaBBB \rightarrow aaabBB \rightarrow aaabbB$

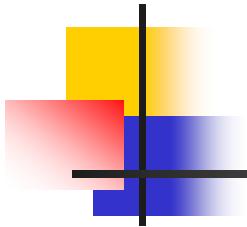
$\rightarrow aaabbb$

$aaaTBB \rightarrow aaaaBBBB \rightarrow aaaabBBB$

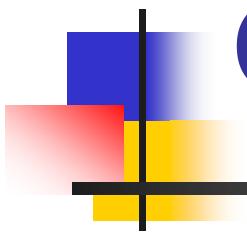
$\rightarrow aaaabbBB$

$\rightarrow aaaabbbB$

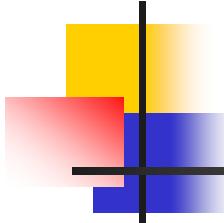
$\rightarrow aaaabbbb$



正則文法と正則言語(再)



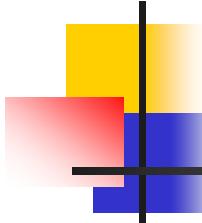
Greibach標準形のCFGとNPDA



DPDAとNPDA

定理 NPDA M が受理する言語 $L(M)$ で, どのような DPDA も $L(M)$ を受理することができないものが存在する.

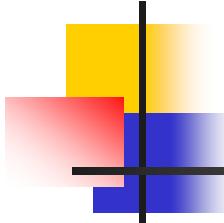
例 $L(M) = \{w(w^R)^k \mid w \in \{a, b\}^*\}$ を受理する DPDA は構成できない.



Greibach標準形からNPDAへの変換

定理 Greibach標準形の文脈自由文法 $G = (N, \Sigma, P, S)$ に対して, NPDA $M = \{\{q\}, N, \Sigma, \delta, q, S, \emptyset\}$ は $L(G) = L(M)$. ただし,

$$\delta(q, c, A) = \{(q, B_1 \dots B_n) \mid A \rightarrow cB_1 \dots B_n \in P\}$$



NPDAからGreibach標準形への変換

定理 空スタックで受理するNPDA $M = \{Q, \Gamma, \Sigma, \delta, q_0, Z_0, \emptyset\}$ に対して、以下のように Greibach標準形の文脈自由文法 $G = (N, \Sigma, P, S)$ を定義すれば $L(G) = L(M)$.

$$N = \{S\} \cup \{[q, c, r] \mid q, r \in Q, c \in \Sigma\}$$

$$P = \{ S \rightarrow [q_0, Z_0, r] \mid r \in Q\}$$

$$\begin{aligned} & \cup \left(\bigcup_{q \in Q, c \in \Sigma, A \in \Gamma} \{ [q, A, r] \rightarrow c[r_1, B_1, r_2][r_2, B_2, r_3] \dots [r_n, B_n, r] \right. \\ & \quad \left. \mid cB_1 \dots B_n \in \delta(q, c, A), r, r_1, \dots, r_n \in Q \} \right) \end{aligned}$$

導出木の例(2)

$N = \{ \text{式} \}$

$\Sigma = \{ a, b, c, +, *, (,) \}$

開始記号 : 式

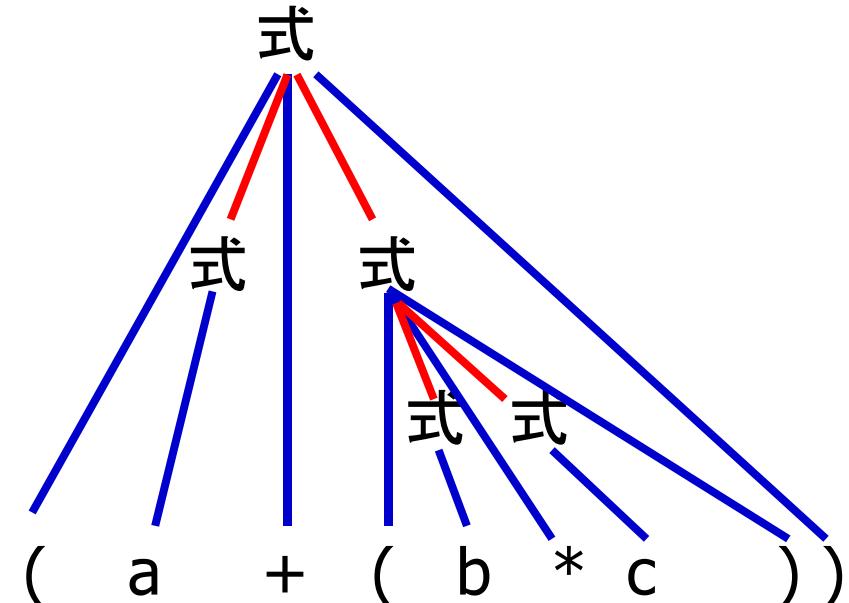
$P = \{ \text{式} \rightarrow (\text{式} + \text{式}),$

$\text{式} \rightarrow (\text{式} * \text{式}),$

$\text{式} \rightarrow a,$

$\text{式} \rightarrow b,$

$\text{式} \rightarrow c \}$

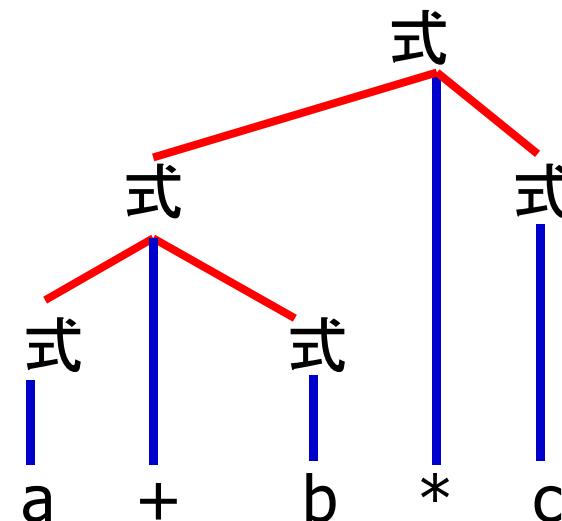
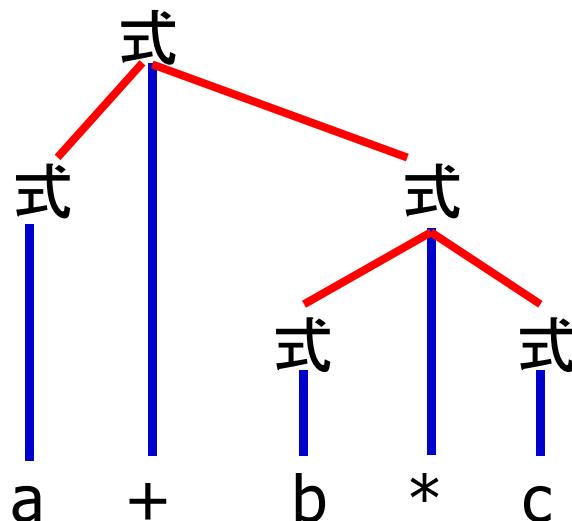


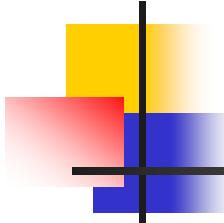
$\text{式} \Rightarrow (\text{式} + \text{式}) \Rightarrow (a + \text{式}) \Rightarrow (a + (\text{式} * \text{式}))$

$\Rightarrow (a + (b * \text{式})) \Rightarrow (a + (b * c))$

曖昧な文法

- ある文法を用いると、一つの単語(記号)の列 σ に対して異なる2種類以上の導出が構成されることがあるとき、その文法は**曖昧である**という。





Greibach標準形の文法

- 文脈自由文法 $G = (N, \Sigma, P, S)$ は、 P 中の全ての生成規則が次の3種類のいずれかであるとき、Greibach標準形であるという。

$$A \rightarrow cB_1 \dots B_n \quad A \rightarrow c \quad S \rightarrow \varepsilon$$

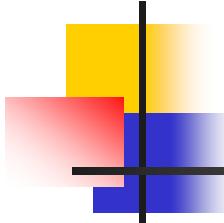
ただし $B_1, \dots, B_n \in N$, $c \in \Sigma$

また $S \rightarrow \varepsilon \in P$ ならば $B_1, \dots, B_n \in N - \{S\}$

例 $G_1'' = (\{S, T, A, B\}, \{a, b\}, P, S)$

$$\begin{aligned} P = \{ & S \rightarrow aT, S \rightarrow aB, T \rightarrow aTB, T \rightarrow aTB, T \rightarrow aBB, \\ & A \rightarrow a, B \rightarrow b \} \end{aligned}$$

$$L(G_1'') = \{a^n b^n \mid n \in \mathbf{N}^+\}$$



最左導出と最右導出

- CFG $G = (N, \Sigma, P, S)$

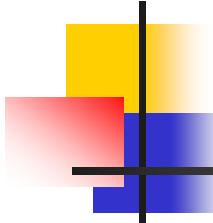
における導出

$$\alpha = \gamma_1 \Rightarrow \gamma_2 \Rightarrow \dots \Rightarrow \gamma_n = \beta$$

において、

$\gamma_1, \gamma_2, \dots, \gamma_n$ の最も左にある非終端記号が導出の対象であるとき、最左導出という

$\gamma_1, \gamma_2, \dots, \gamma_n$ の最も右にある非終端記号が導出の対象であるとき、最右導出という



Greibach標準形による構文解析

$$P = \{S \rightarrow aT, S \rightarrow aB, T \rightarrow aTB, T \rightarrow aBB, A \rightarrow a, B \rightarrow b\}$$

aaaabbbb

$S \rightarrow aB \rightarrow ab$

$aT \rightarrow aaBB \rightarrow aabB \rightarrow aabb$

$aaTB \rightarrow aaaBBB \rightarrow aaabBB \rightarrow aaabbB$

$\rightarrow aaabbb$

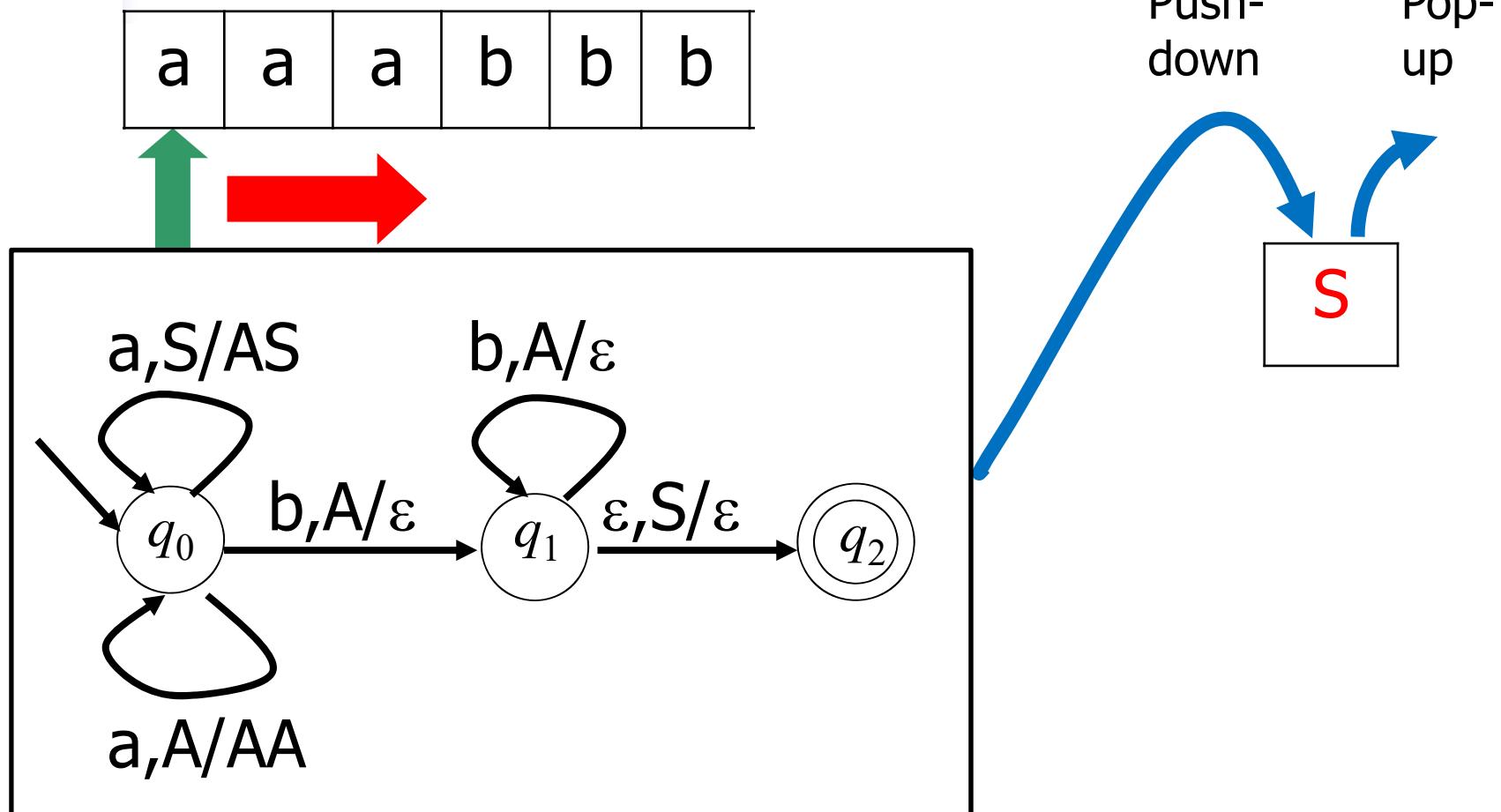
$aaaTBB \rightarrow aaaaBBBB \rightarrow aaaabBBB$

$\rightarrow aaaabbBB$

$\rightarrow aaaabbbB$

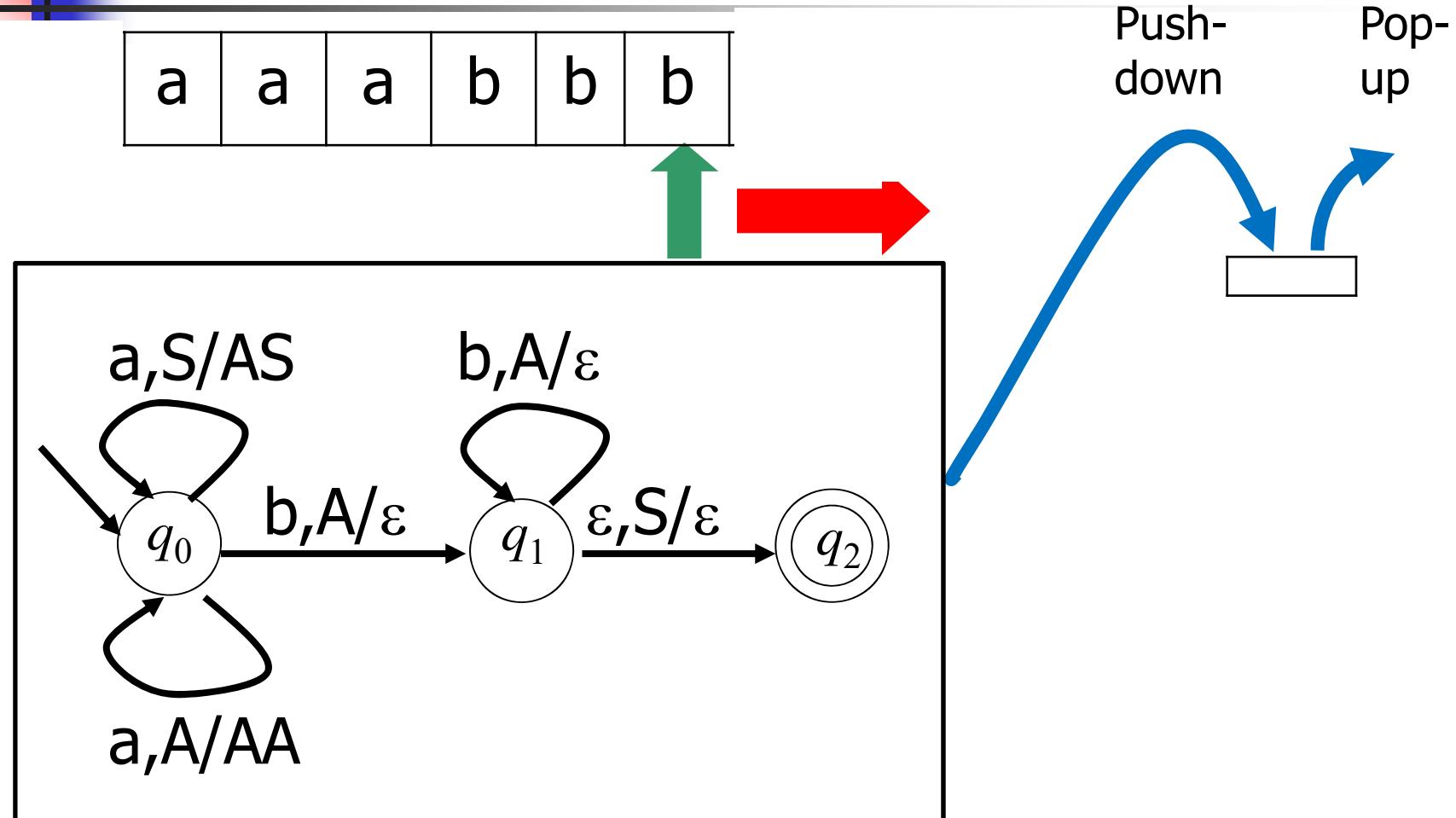
$\rightarrow aaaabbbb$

プッシュダウン・オートマトン(例1)

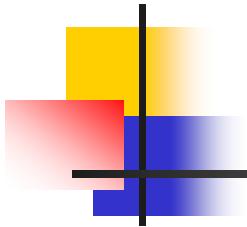


(q_0, S)

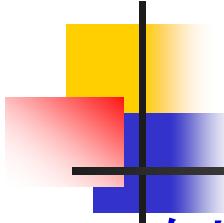
プッシュダウン・オートマトン(例8)



$$\begin{aligned}
 (q_0, S) &\Rightarrow (q_0, AS) \Rightarrow (q_0, AAS) \Rightarrow (q_0, AAAS) \\
 &\Rightarrow (q_1, AAS) \Rightarrow (q_1, AS) \Rightarrow (q_1, S) \Rightarrow (q_2, \varepsilon)
 \end{aligned}$$



下降型解析と上昇型解析



導出に決定性を持つ文法の例

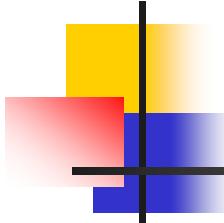
例 $G_1''' = (\{S, T, A, B\}, \{a, b\}, P, S)$

$P = \{S \rightarrow aT, T \rightarrow aTB, T \rightarrow b, B \rightarrow b\}$

$L(G_1''') = \{a^n b^n \mid n \in \mathbf{N}^+\}$

aaaabbbbの最左導出

$S \Rightarrow aT \Rightarrow aaTB \Rightarrow aaaTBBB$
aaaabbbb **aaaabbbb** **aaaabbbb** **aaaabbbb**
 $\Rightarrow aaaaTBBB \Rightarrow aaaabBBB \Rightarrow aaaabbBB$
 aaaabbbb **aaaabb**bb**** **aaaabb**bb****
 $\Rightarrow aaaabbbB \Rightarrow aaaabbbb$
 aaaabbbb



下降型解析とLL(k)文法

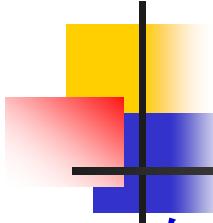
- 開始記号 S から語 w へ解析を進める方法を下降型解析という
- 文法 $G = (N, \Sigma, P, S)$ は、最左導出により

$$S \Rightarrow \dots \Rightarrow uA\beta \quad (u \in \Sigma^*, A \in N, \beta \in (\Sigma \cup N)^*)$$

と導出され、さらに

$$uA\beta \Rightarrow u\alpha\beta \Rightarrow \dots \Rightarrow uv \quad (v \in \Sigma^*, \alpha \in (\Sigma \cup N)^*)$$

と導出されるとき、 v の先頭から高々 k 記号を見るだけで A に適用すべき生成規則 $A \rightarrow \alpha$ が一意に定まるとき LL(k) 文法という



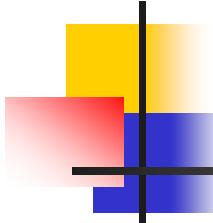
LL(2)文法の例

例 $G_2 = (\{S, A, B\}, \{a, b, c\}, P, S)$

$P = \{S \rightarrow aBc, S \rightarrow bAc, A \rightarrow aaB, A \rightarrow abA, A \rightarrow a, B \rightarrow bbA, B \rightarrow baB, B \rightarrow b\}$

abbacの最左導出

S	$\Rightarrow aBc$	$\Rightarrow abbAc$	$\Rightarrow abbac$
abbac	abbac	abbac	abbac



上昇型解析とLR(k)文法

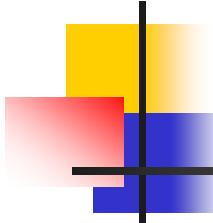
- 語 w から始めて、生成規則 $A \rightarrow \alpha$ の右辺 α に一致するようなもの(ハンドル)を左辺 A に置き換える(還元する)ことで S まで還元する方法を上昇型解析という
 - CKY法は上昇型解析の一種である
- 文法 $G = (N, \Sigma, P, S)$ は、 S が生成規則の右辺に現れず、最右導出により

$$\alpha\beta\nu \Rightarrow \dots \Rightarrow uv \quad (u, v \in \Sigma^*, \alpha, \beta \in (\Sigma \cup N)^*)$$

と導出され、さらに

$$S \Rightarrow \dots \Rightarrow \alpha Av \Rightarrow \alpha\beta\nu \quad (A \in N)$$

と導出されるとき、 v の先頭から高々 k 記号を見るだけで β がハンドルであることがわかり、生成規則 $A \rightarrow \beta$ が一意に定まるとき LR(k) 文法という



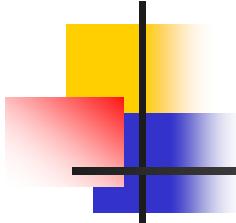
LR(1)文法の例

$$G_3 = (N = \{S\}, \Sigma = \{ a, +, *, (,), \# \}, P, S)$$

$$\begin{aligned}P = & \{ S \rightarrow E\#, E \rightarrow E + T, E \rightarrow T, T \rightarrow T * F, \\& T \rightarrow F, F \rightarrow (E), F \rightarrow a \}\end{aligned}$$

(a*a+a)#の最右導出

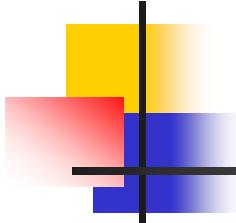
$$\begin{aligned}S &\Rightarrow E\# \Rightarrow T\# \Rightarrow F\# \Rightarrow (E)\# \Rightarrow (E+T)\# \\&\Rightarrow (E+F)\# \Rightarrow (E+a)\# \Rightarrow (T+a)\# \\&\Rightarrow (T^*F+a)\# \Rightarrow (T^*a+a)\# \Rightarrow (F^*a+a)\# \\&\Rightarrow (a^*a+a)\#\end{aligned}$$



上昇解析の実際(1)

$$P = \{S \rightarrow E\#, E \rightarrow E + T, E \rightarrow T, T \rightarrow T * F, \\ T \rightarrow F, F \rightarrow (E), F \rightarrow a\}$$

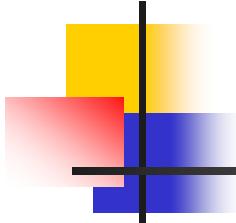
入力バッファ	スタック	次の動作
(a * a + a) #		shift
a * a + a) #	(shift
* a + a) #	(a	$F \rightarrow a$ による還元
* a + a) #	(F	$T \rightarrow F$ による還元
* a + a) #	(T	先読み
* a + a) #	(T	shift ($E \rightarrow T$ による還元ではなく)
a + a) #	(T *	



上昇解析の実際(2)

$$P = \{S \rightarrow E\#, E \rightarrow E + T, E \rightarrow T, T \rightarrow T * F, \\ T \rightarrow F, F \rightarrow (E), F \rightarrow a\}$$

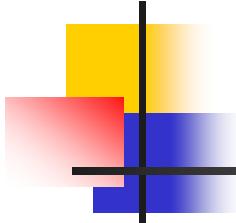
入力バッファ	スタック	次の動作
+ a)<#	(T *a	$F \rightarrow a$ による還元
+ a)<#	(T *F	先読み
+ a)<#	(T	$T \rightarrow T * F$ による還元
+ a)<#	(T	$E \rightarrow T$
+ a)<#	(E	shift
a)<#	(E+	shift
)#	(E+a	shift



上昇解析の実際(3)

$$P = \{S \rightarrow E\#, E \rightarrow E + T, E \rightarrow T, T \rightarrow T * F, \\ T \rightarrow F, F \rightarrow (E), F \rightarrow a\}$$

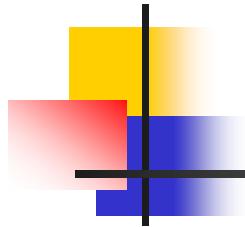
入力バッファ	スタック	次の動作
)#	(E + a	$F \rightarrow a$ による還元
)#	(E+F	$T \rightarrow F$
)#	(E+T	先読み
)#	(E	$E \rightarrow E+T$ による還元
)#	(E	shift
#	(E)	shift
	(E) #	$F \rightarrow (E)$ による還元



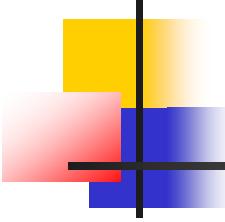
上昇解析の実際(4)

$$P = \{S \rightarrow E\#, E \rightarrow E + T, E \rightarrow T, T \rightarrow T * F, \\ T \rightarrow F, F \rightarrow (E), F \rightarrow a\}$$

入力バッファ	スタック	次の動作
#	T	$E \rightarrow T$ による還元
#	E	shift
	$E\#$	$S \rightarrow E\#$ による還元
	S	受理



文脈自由言語の特徴

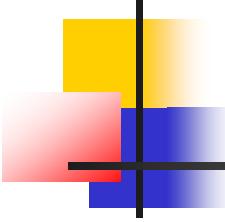


正則言語の特徴

Pumping Lemma 要素数が有限ではない正則言語 L に対して、自然数 N があって、 $s \in L$ かつ $|s| \geq N$ なるすべての語 s は $s=uvw$ かつ $|w| \geq 1$ かつすべての $i \geq 1$ に対して $uv^iw \in L$ が成り立つような部分語 w を含んでいる。

- N は L に依存して定まる(L が変わると N は変わる)
- 実は N は $L=L(M)$ なる(状態数最小の)DFA M の状態数+1とすればよい。

証明のアイデア $L=L(M)$ なる(状態数最小の)DFA $M = (\Sigma, S, d, q_0, F)$ の状態数を N' とする。 L は要素数が有限ではないので必ず L には $|s| \geq N'+1$ を満たす記号列を含む。すると M が s を受理する過程 $q_0 \Rightarrow q_1 \Rightarrow \dots \Rightarrow q_{|s|}$ を考えると鳩の巣原理から $q_i=q_j$ が成り立つ q_i, q_j ($i \neq j$)があるはず。



正則言語ではない言語

$$G_1 = (N = \{S\}, \Sigma = \{a, b\}, P = \{S \rightarrow ab, S \rightarrow aSb\}, S)$$

$$L(G_1) = \{w \in \{a, b\}^* \mid a^n b^n, n \in \mathbf{N}^+ \}$$

$L(G_1)$ は正則言語ではない。

($\Leftrightarrow L(G_1)$ を受理するDFAを構成することはできない)

証明の方針 (背理法を用いる) $L(G_1) = L(M)$ を満たす

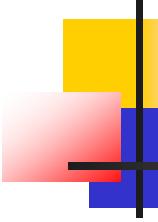
DFA M が構成できたと仮定し、その状態数を N' とする。

このとき $n = N'/2 + 1$ とおくと $|a^n b^n| \geq N' + 1$ だから

$a^n b^n = uvw$ かつ $|w| \geq 1$ かつ

すべての $i \geq 1$ に対して $uv^i w \in L$

と分解できるはずである。このとき、 $uv^2 w \notin L$ であること
を $|u|$ と $|w|$ で場合分けして示す。



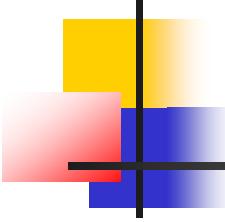
文脈自由言語の特徴

Pumping Lemma 要素数が有限ではない文脈自由言語 L に対して、自然数 N があって、 $s \in L$ かつ $|s| \geq N$ なるすべての語 s は $s=uvwxy$ かつ $|vx| \geq 1$ かつ すべての $i \geq 1$ にに対して $uv^iwx^i y \in L$ が成り立つように分解できる。

- N は L に依存して定まる(L が変わると N は変わる)

証明のアイデア $L - \{\varepsilon\} = L(G)$ なる Chomsky 標準形の CFG $G = (N, \Sigma, P, S)$ について、 $|N| = n$ とする。語 s の構文木の高さ(根から葉までの経路の最大値)の最大値(構文木は複数構成できるかもしれない)を $h+1$ とすると、 G が Chomsky 標準形であることから、 $|s| \leq 2^h$ である。

L の要素数が有限ではないので必ず L には $|s| \geq 2^n$ を満たす語が含まれている。



文脈自由言語の特徴(続き)

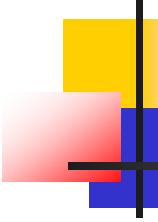
$|s| \geq 2^n$ を満たす語の構文木の高さは $n+1$ 以上であるから、 $n+1$ 個以上の内部接点を持つ経路が含まれている。すると鳩の巣原理から、この経路上に2度以上現れる非終端記号 A がある。この経路の導出は(直接導出の順序を入れ替えれば)

$$S \Rightarrow \dots \Rightarrow uAy \Rightarrow \dots \Rightarrow uvAxy \Rightarrow \dots \Rightarrow uvwxy$$

となっているはずであるから、後半部分は

$$\Rightarrow uvAxy \Rightarrow \dots \Rightarrow uv^iAx^i y \Rightarrow \dots \Rightarrow uv^iwx^i y$$

とできる。このような A で $|vwx| \leq 2^n$ を満たすものが存在することは背理法による。



文脈自由言語ではない言語(1)

$$L = \{a^n b^n c^n \mid n \in \mathbb{N}^+ \}$$

L は文脈自由言語ではない.

($\Leftrightarrow L$ を受理するCFGを構成することはできない)

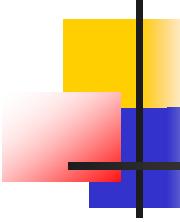
証明の方針 (背理法を用いる) $L - \{\varepsilon\} = L(G)$ を満たす Chomsky標準形のCFG G が構成できたと仮定し, その非終端記号数を m とする. このとき

$|a^n b^n c^n| \geq 2^m$ については

$a^n b^n c^n = uvwxy$ かつ $|w| \geq 1$ かつ

すべての $i \geq 1$ に対して $uv^iwx^i y \in L$

と分解できるはずである. このとき, $uv^2wx^2y \notin L$ であることを場合分けして示す.



文脈自由言語と集合演算

- L, M を文脈自由言語とするとき

$L \cup M, LM, L^*$ は文脈自由言語となるが

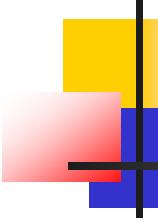
$L \cap M, \Sigma^* - L$ は必ずしも文脈自由言語とならない

しかし M が正則言語であれば $L \cap M$ は文脈自由となる

例 $L = \{a^n b^n c^k \mid n \in \mathbf{N}^+, k \in \mathbf{N}^+ \}$

$$M = \{a^k b^n c^n \mid n \in \mathbf{N}^+, k \in \mathbf{N}^+ \}$$

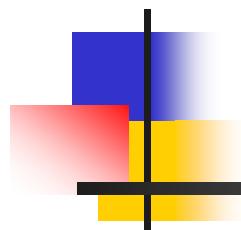
とすると, $L \cap M = \{a^n b^n c^n \mid n \in \mathbf{N}^+ \}$



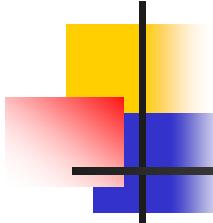
等価性の判定

- 任意のCFG G_1, G_2 に対して, $L(G_1) = L(G_2)$ であるかどうかを判定するアルゴリズムを構成することはできない.
 - “ $L(G_1) = \Sigma^*$ であるかどうかを判定するアルゴリズムを構成することはできない”の系である
- DPDA M_1, M_2 に対して, $L(M_1) = L(M_2)$ であるかどうかを判定するアルゴリズムを構成することができる.
 - 後者は2001年になってSénizerguesによってようやく証明された

言語・オートマトン Turing機械と決定可能性



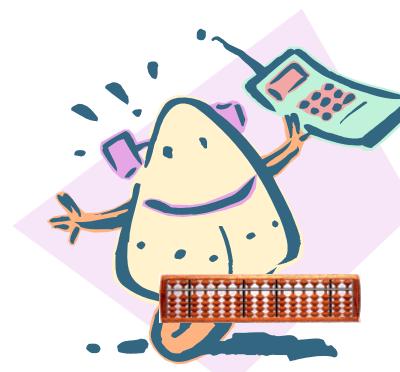
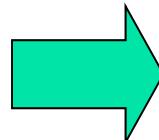
山本章博
情報学研究科 知能情報学専攻
(工学部 情報学科)
2017年度版



“計算”を人工的に作る

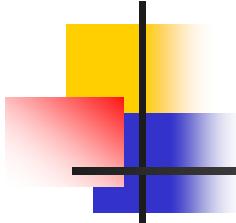
- コンピューター(computer)は、人間が行う“計算”という行為をシミュレートする研究がもとになっている。

$$(((2 \times 3) + (4 \times 6)) + (7 \times 8)) \div (2 + 4)$$



computer (計算手)

computer (計算機)



“計算”に関する視点

- 19世紀末から20世紀初頭にかけて、数学の世界における“計算という行為”に関する問題意識の萌芽
 - 特にHilbert学派
 - 数学基礎論の成立
- 自動機械(automaton)



2012 THE ALAN TURING YEAR

A Centenary Celebration of the Life and Work of Alan Turing

Centenary Events

- [ATY EVENTS OVERVIEW](#)
- [ATY EVENTS CALENDAR](#)
- [ATY EVENTS A4 HANDOUT](#)
- [ATY RESOURCES](#)
- [TCAC Arts & Culture Subtee](#)
- [TCAC Media Group](#)
- [Turing Manchester 2012](#)
- [TCAC Manchester](#)
- [Alan Turing Jahr 2012](#)
- [TCAC Germany](#)
- [Alan Turing Jaar 2012](#)
- [AAAI Turing Lecture New](#)
- [ACE 2012, Cambridge](#)
- [ACM Centenary Celebration](#)
- [AI at Donetsk, Ukraine](#)
- [AI*IA Symp. Artificial Intelligence](#)
- [Alan Mathison Turing, Roma](#)
- [Alan Turing Centenary in Calgary](#)
- [ALAN TURING CONF, Manchester](#)
- [Alan Turing Days in Lausanne](#)
- [AMS Special Session, USA](#)
- [AMS-ASL Joint Math Meeting](#)
- [Animation12, Manchester](#)
- [BBC Television Documentary New](#)

• To link to this webpage please use the url: <http://www.turingcentenary.eu/> - and add the ATY logo (suitably resized) to your webpage. See also [pdf version](#) or [monochrome version](#)

• If you wish to be included in the Turing Centenary email list, please enter your email address here and press Submit:

The Alan Turing Year on Facebook - and on Twitter

ATY Press and Media Contact - [Daniela Derbyshire](#) - email: turing @ live.co.uk



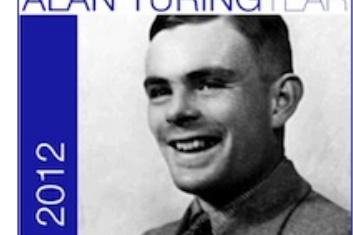
The Turing Test
a one-hour opera, sung in English
UK tour 2012 - help us make it happen!



June 23, 2012, is the Centenary of Alan Turing's birth in London. During his relatively brief life, Turing made a unique impact on the history of computing, computer science, artificial intelligence, developmental biology, and the mathematical theory of computability.

ALAN TURINGYEAR

2012

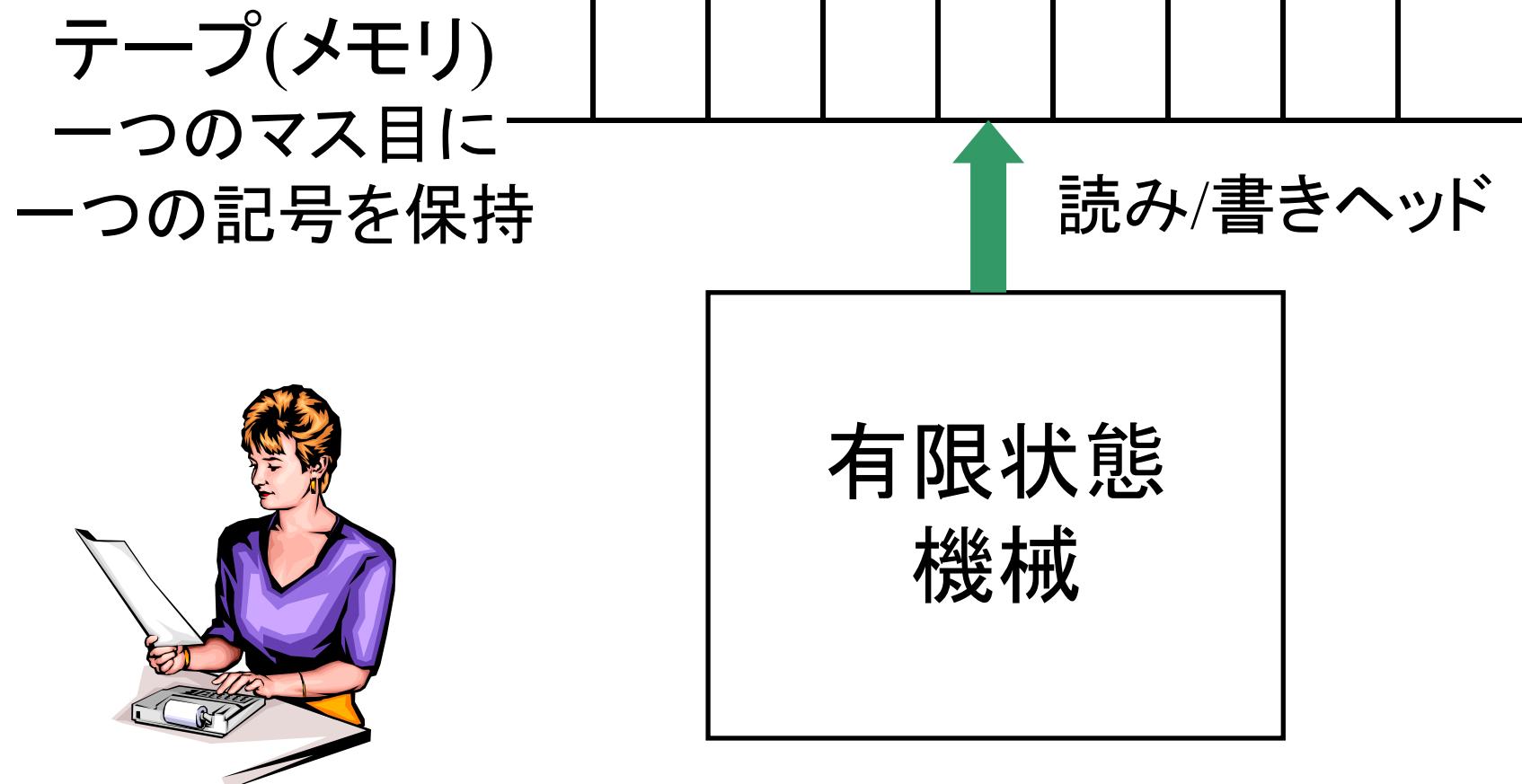


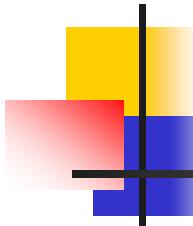
News

- 19.04.12**
GCHQ releases two codebreaking papers by Alan Turing
- 09.04.12**
The biography of Alan M Turing by his mother Sara appears
- 06.04.12**
Manchester Pride Festival to honour Alan Turing

Turing Machine(1936)

Alan Turingが考案



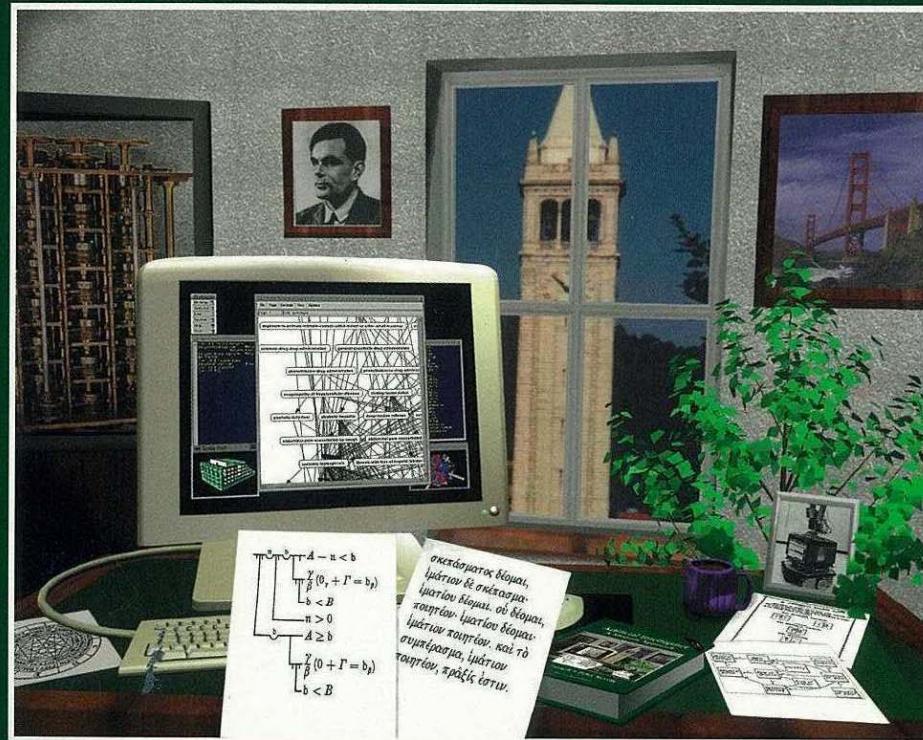


Artificial Intelligence

A Modern Approach

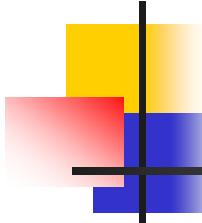
SECOND EDITION

The Intelligent
Agent Book



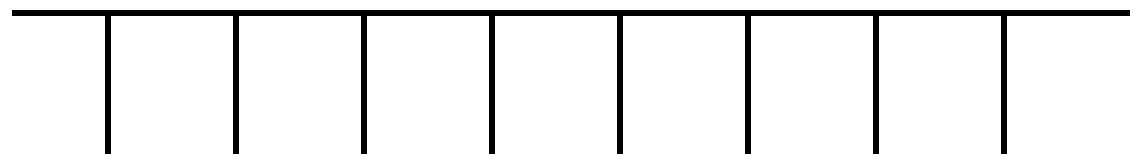
Stuart Russell • Peter Norvig

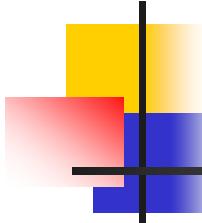
Prentice Hall Series in Artificial Intelligence



Turingによる計算の機械化(1)

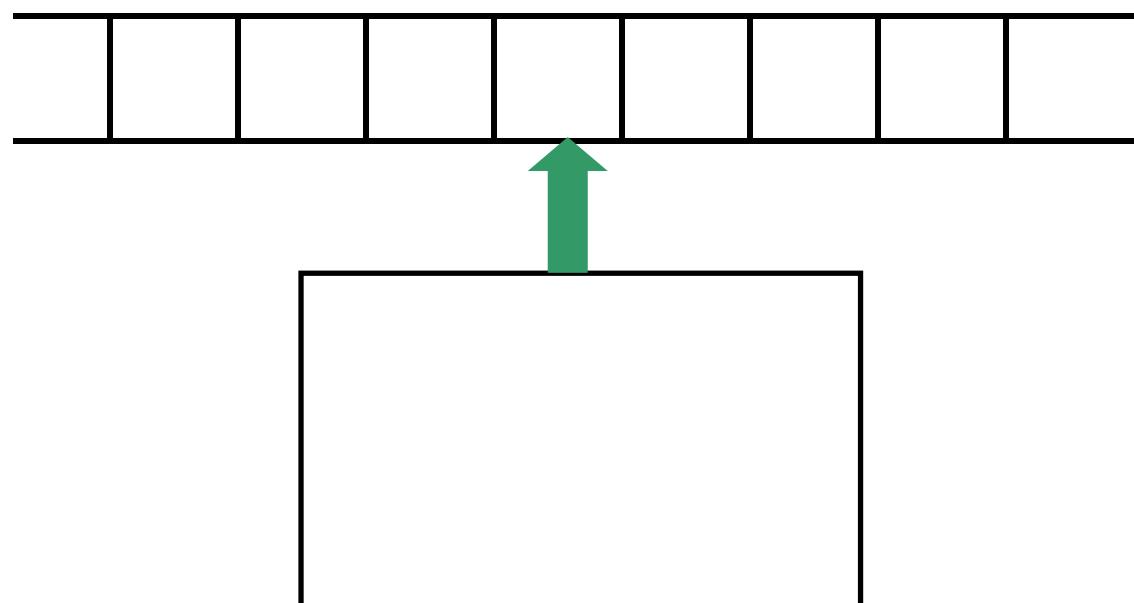
- Computing is normally done by writing certain symbols on paper. We may suppose this paper is divided into squares like a child's arithmetic book.
...
- I assume then that the computation is carried out on one-dimensional paper, i.e. on a tape divided into squares.

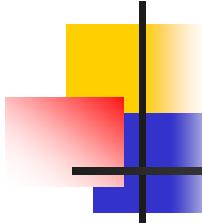




Turingによる計算の機械化(2)

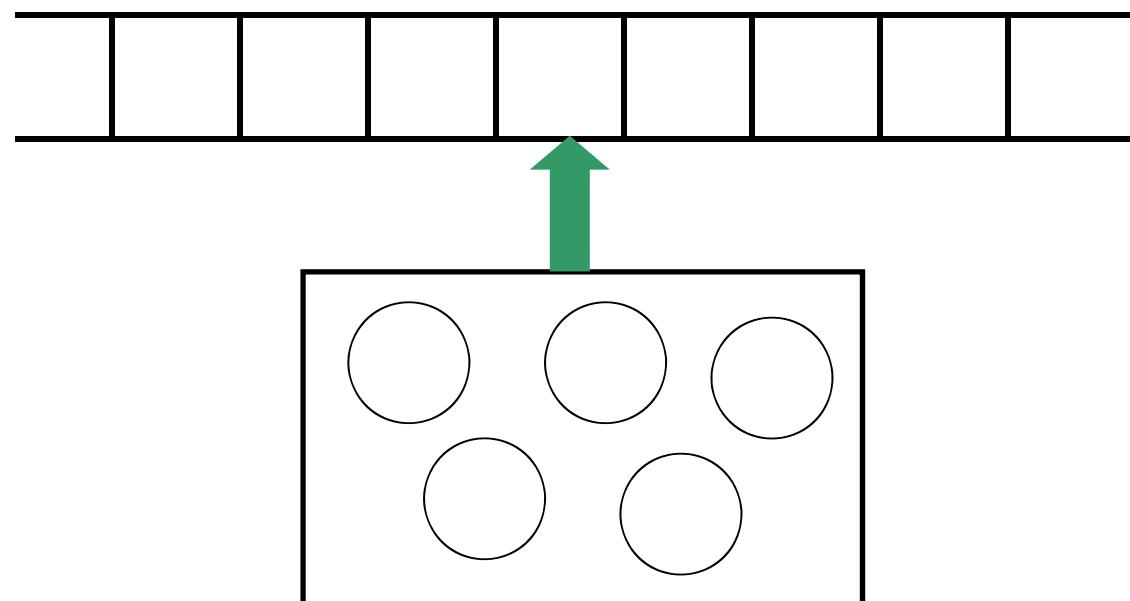
- The behaviour of the computer at any moment is determined by the symbols which he is observing and his “state of mind” at that moment.

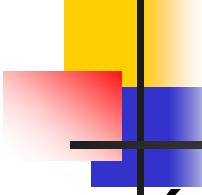




Turingによる計算の機械化(3)

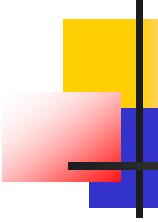
- We will also suppose that the number of states of mind which need be taken into account is finite.





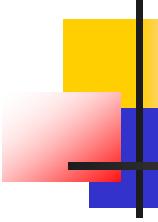
Turingによる計算の機械化(4)

- (a) Changes of the symbol on one of the observed squares.
 - (b) Changes of one of the squares observed to another square within L squares of one of the previously observed squares.
-
- A. A possible change (a) of symbol together with a possible change of state of mind.
 - B. A possible change (b) of observed squares, together with a possible change of state of mind.



集合と関数によるTMの表現

- 以下のような構成要素からなる組 $M = (\Gamma, \Sigma, S, \delta, s_0, B, F)$ を(決定性)Turing機械とよぶ.
 - Γ はアルファベット: テープ記号の集合
 - $\Sigma \subset \Gamma$ はアルファベット: 入力記号
 - S は空でない有限集合であり,
その要素を状態とよぶ.
 - δ は $S \times \Gamma \rightarrow S \times \Gamma \times \{L, R\}$ なる関数
 - $s_0 \in S$ は特定の状態であり, 初期状態とよぶ.
 - $B \in \Gamma - \Sigma$ は特定の記号であり, 空白記号とよぶ
 - $F \subset S$ は特定の状態の集合であり, その要素を終了状態, もしくは受理状態とよぶ.



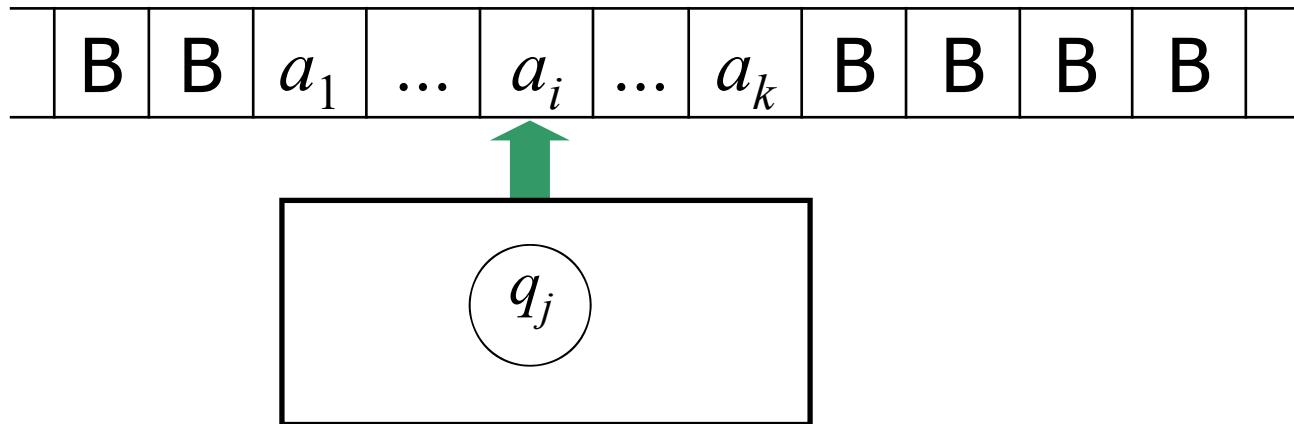
状態遷移表による表現

- テープ記号: B, c_1, \dots, c_n
- 状態: $q_0, q_1, q_2, \dots, q_m$
- ヘッドを動かす方向 : $D = L$ or R
- 遷移関数 $\delta(q_i, c_j) = \langle q_k, c_h, D \rangle$
 - 引数の組 (q_i, c_j) によっては値が定義されない
⇒ 関数としては特別な値 λ (あるいは \perp) を返すとする
機械としては特別な状態に遷移すると解釈する

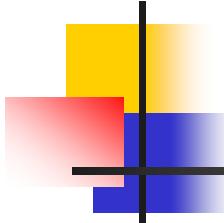
	F	B	c_1	...	c_n
q_0					
q_1					
...		
q_m					

計算状況(時点表示)

- 無限長のテープの有限部分のみ空白でない記号が記入されている $a_1 \dots q_j a_i \dots a_k$
- 時点表示 $\sigma = (q_j, a_1 \dots a_i \dots a_k, i)$ または $a_1 \dots q_j a_i \dots a_k$

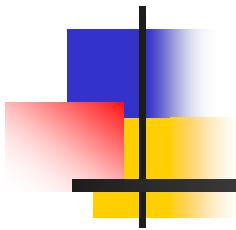


- $(q_j, a_1 \dots a_i \dots a_k, i) \Rightarrow (q_k, a_1 \dots a_l \dots a_k, l)$
遷移表を1回だけ用いて $\sigma = (q_j, a_1 \dots a_i \dots a_k, i)$ が
 $\tau = (q_h, a_1 \dots a_l \dots a_k, l)$ ($l = \pm i$) に遷移する



終端計算状況

- 計算状況 $(q_h, c_1 \dots c_i \dots c_k, i)$ が終端計算状況
 $\Leftrightarrow \delta(q_h, c_i)$ が定義されていない



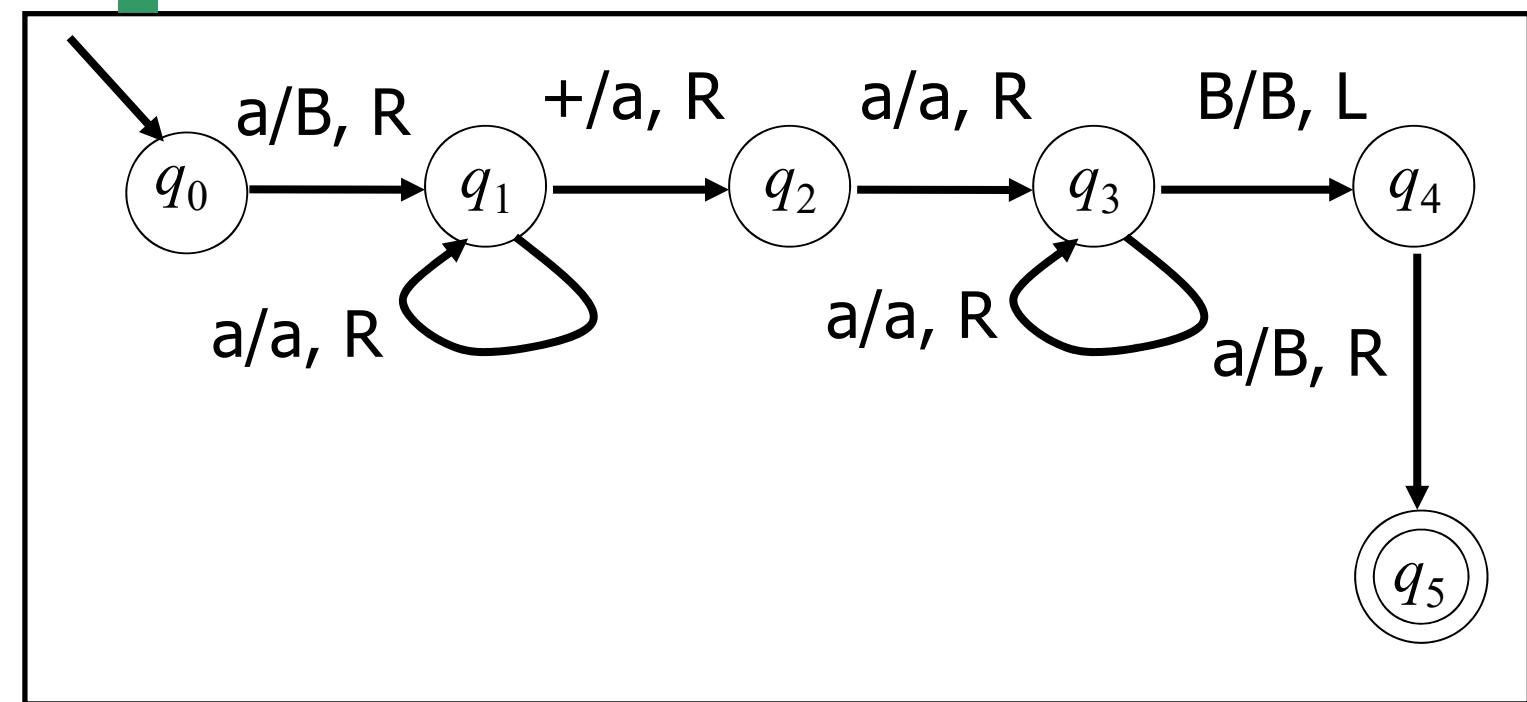
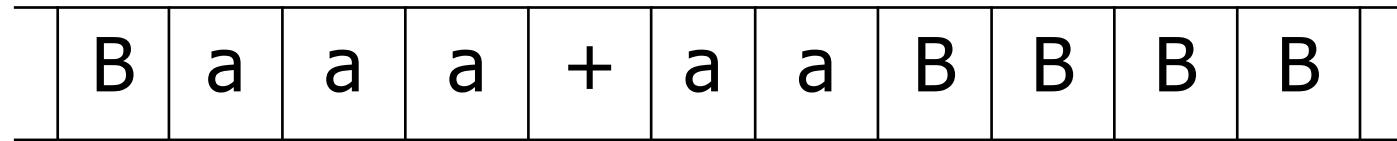
計算モデルとしてのTuring機械

1進表現による加算のためのTM

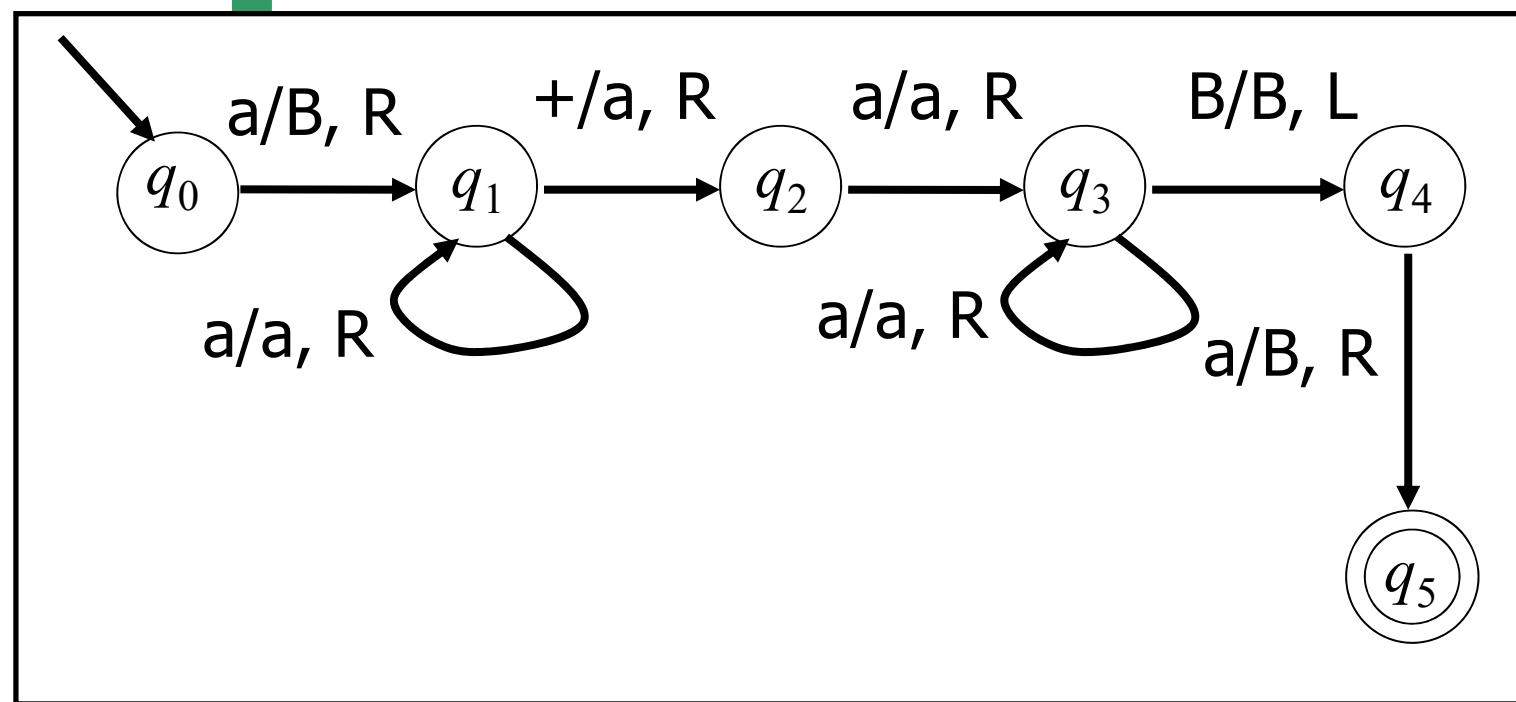
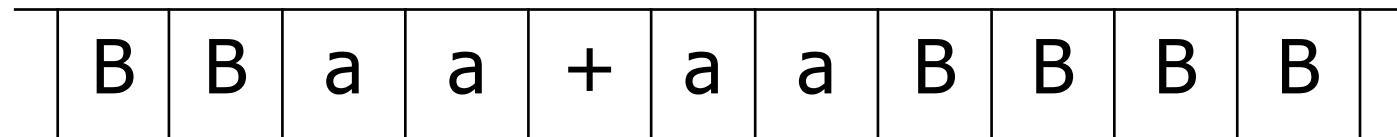
- テープ記号: B, a, + 入力記号: a, +
 - 自然数の表現: 1進法 $0 \rightarrow a, 1 \rightarrow aa, 2 \rightarrow aaa, \dots$
- 状態: q_0, q_1, \dots, q_7 終了状態: q_7
- 状態遷移表:

	F	B	a	+
q_0		λ	$\langle q_1, B, R \rangle$	λ
q_1		λ	$\langle q_1, a, R \rangle$	$\langle q_2, a, R \rangle$
q_2		λ	$\langle q_3, a, R \rangle$	λ
q_3		$\langle q_4, B, L \rangle$	$\langle q_3, a, R \rangle$	λ
q_4		λ	$\langle q_5, B, R \rangle$	λ
q_5	○	λ	λ	λ

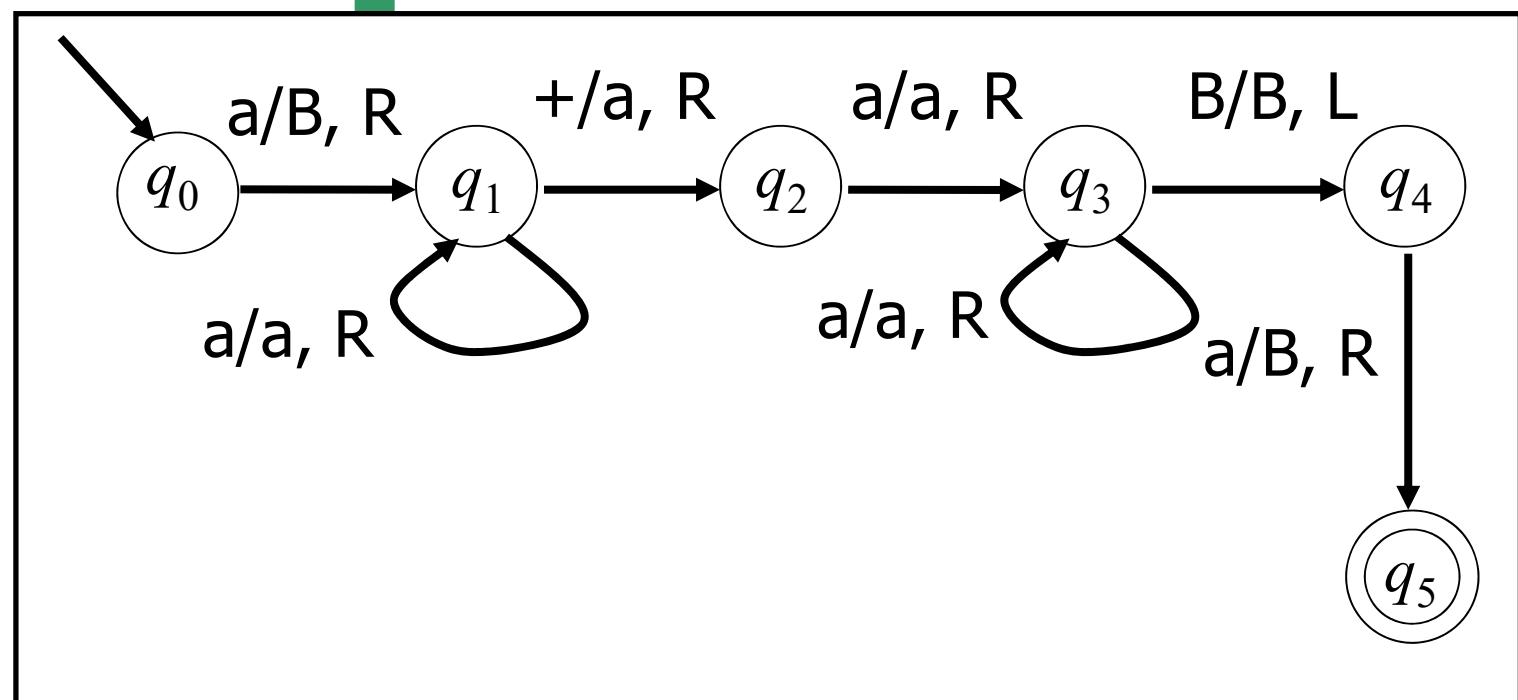
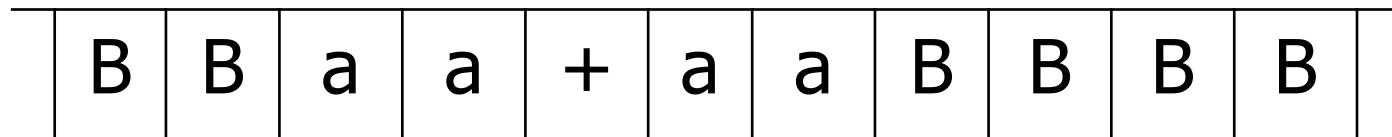
1進表現による加算のためのTM(1)



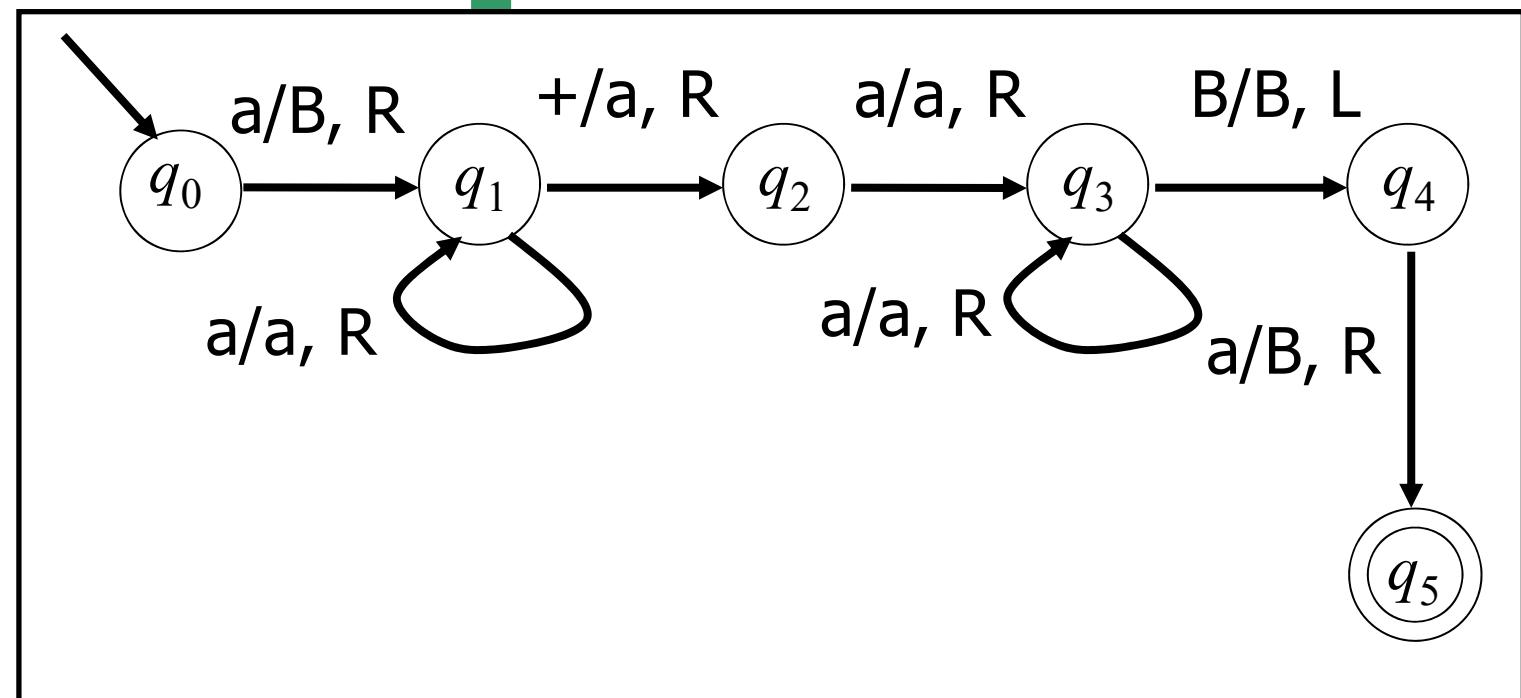
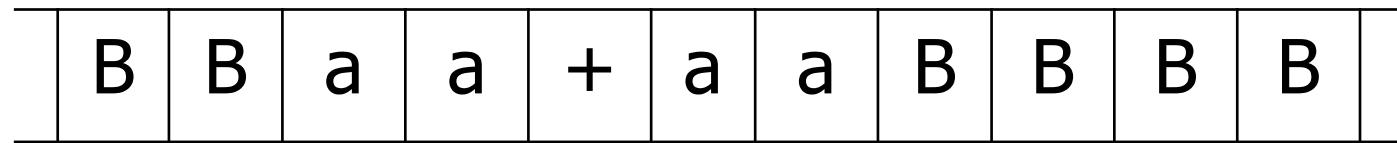
1進表現による加算のためのTM(2)



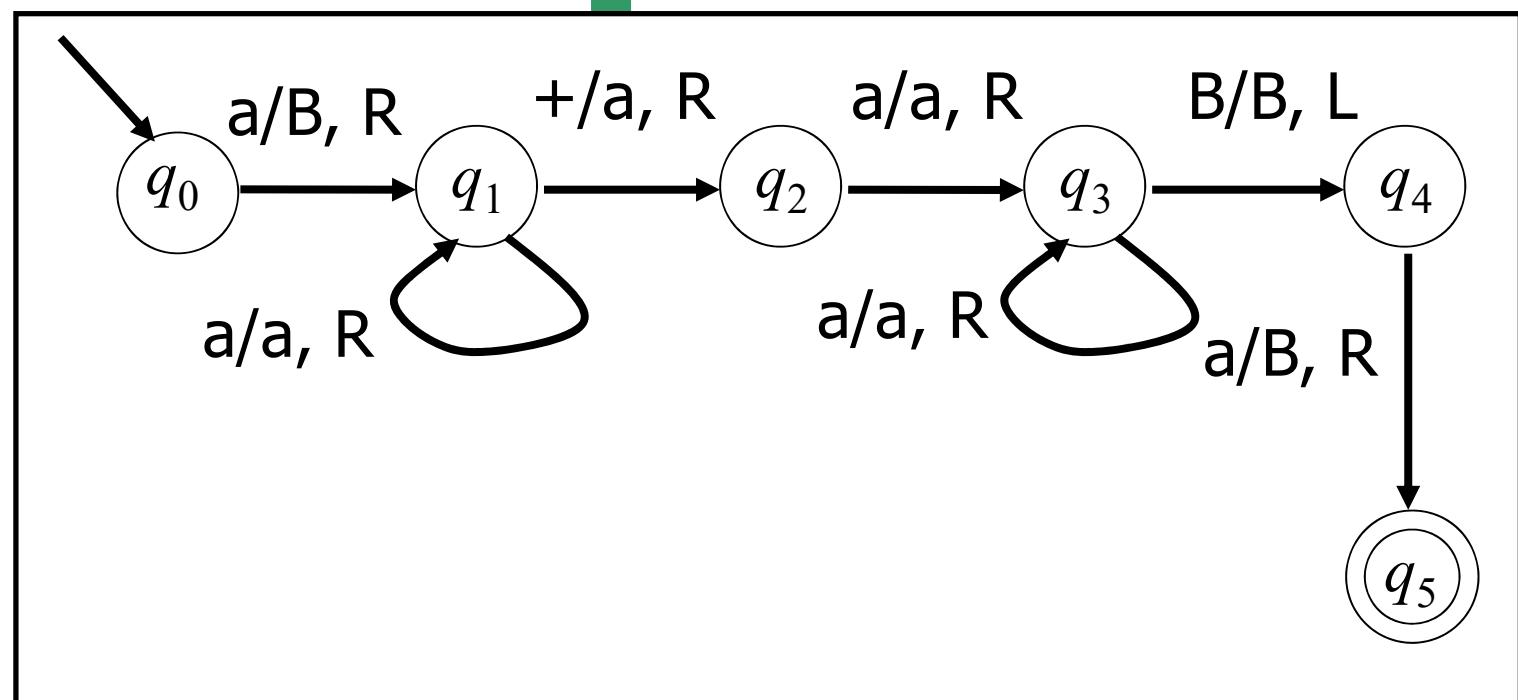
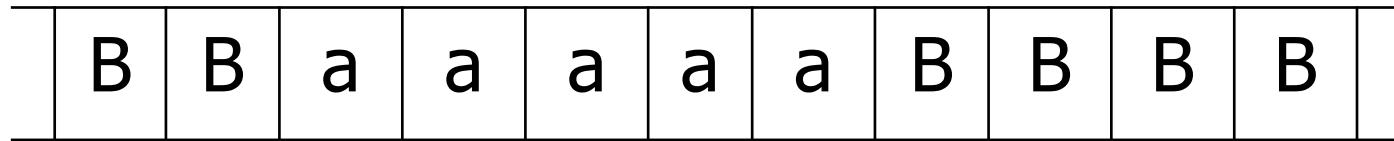
1進表現による加算のためのTM(3)



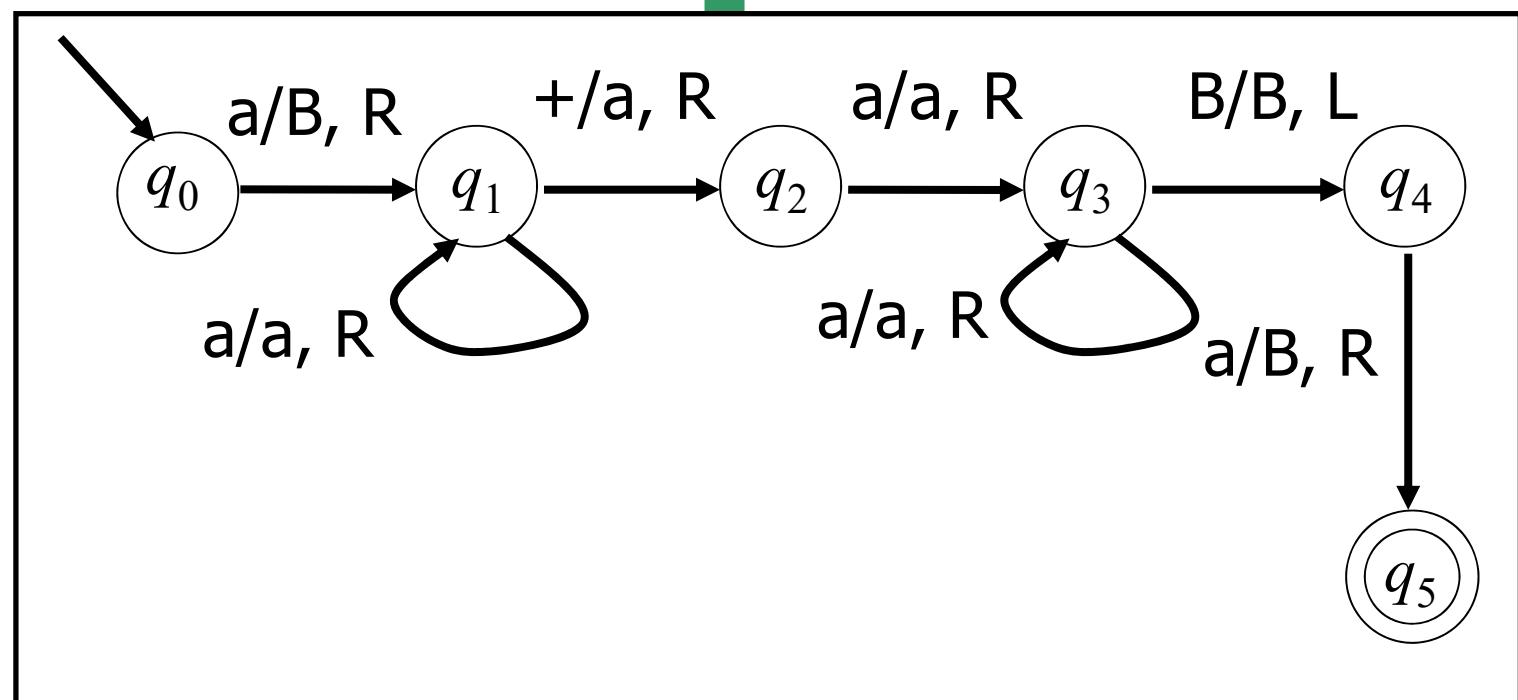
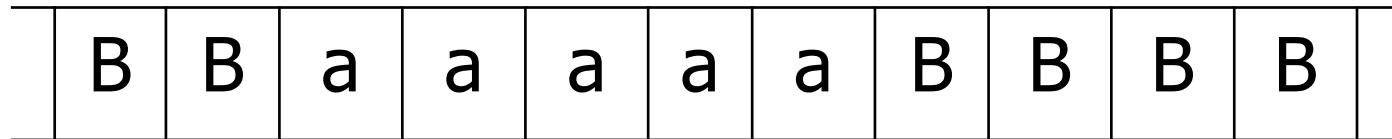
1進表現による加算のためのTM(4)



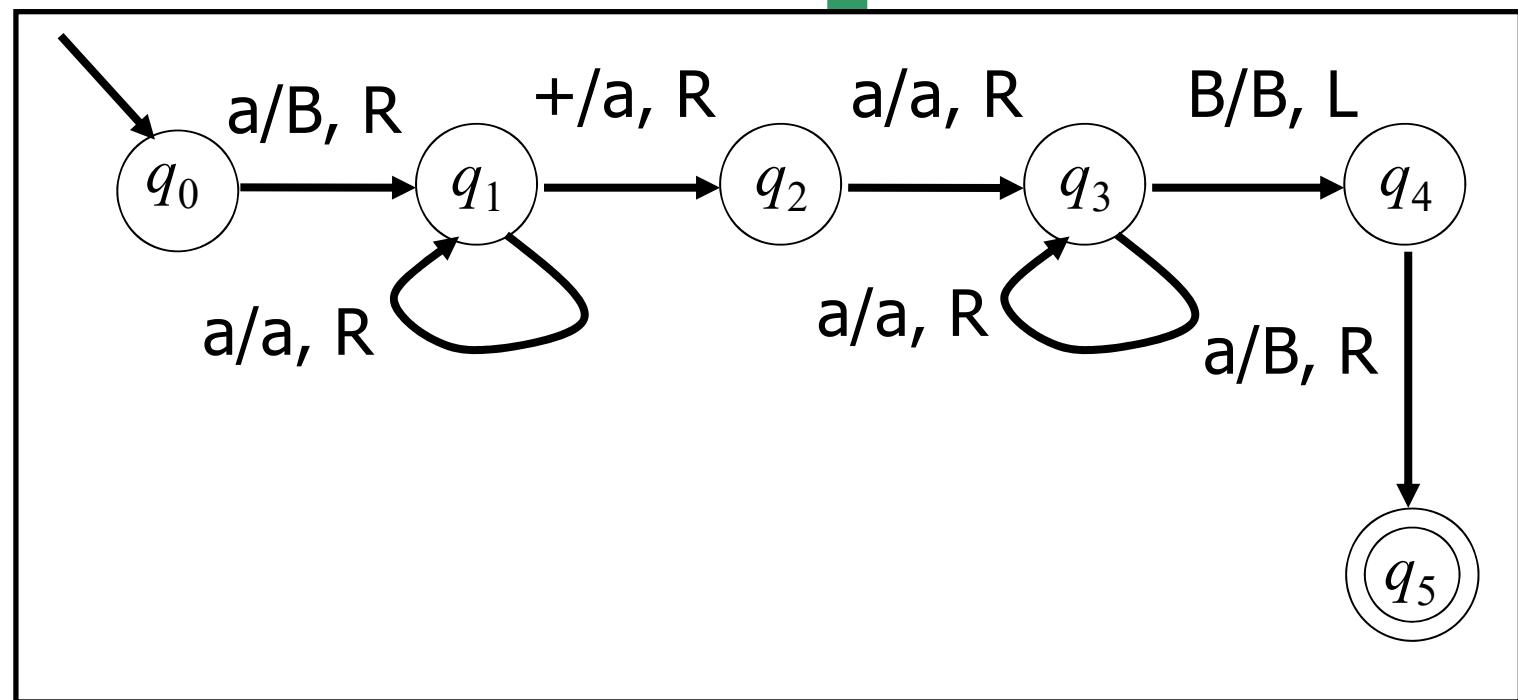
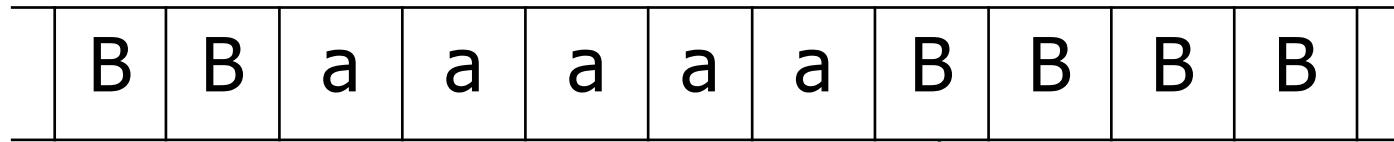
1進表現による加算のためのTM(5)



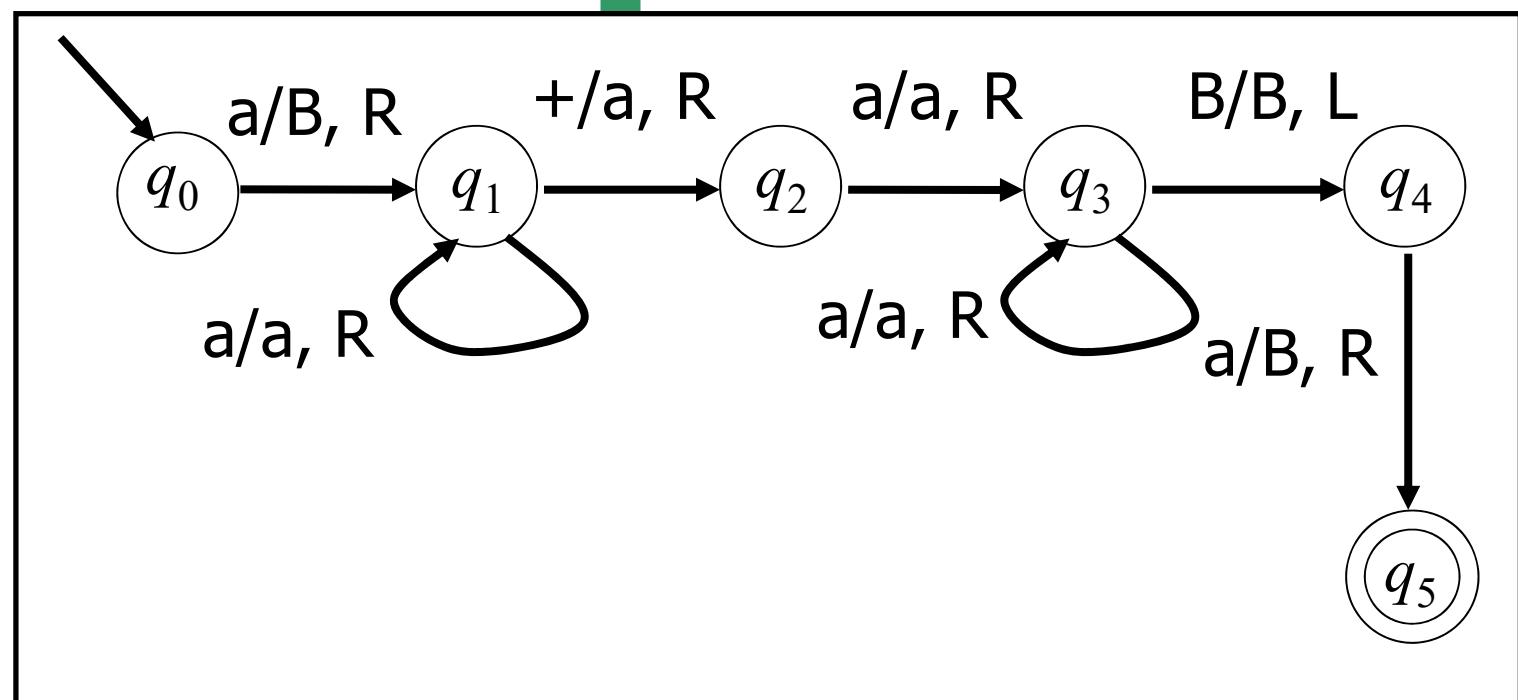
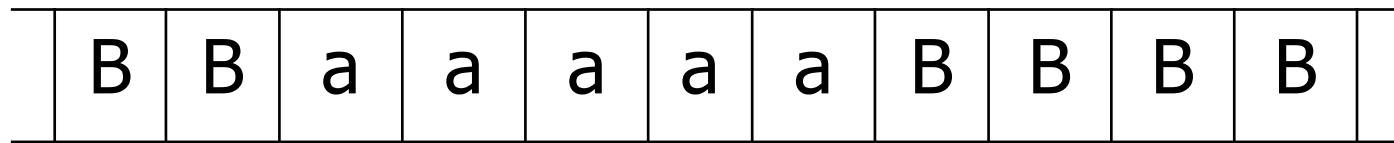
1進表現による加算のためのTM(7)



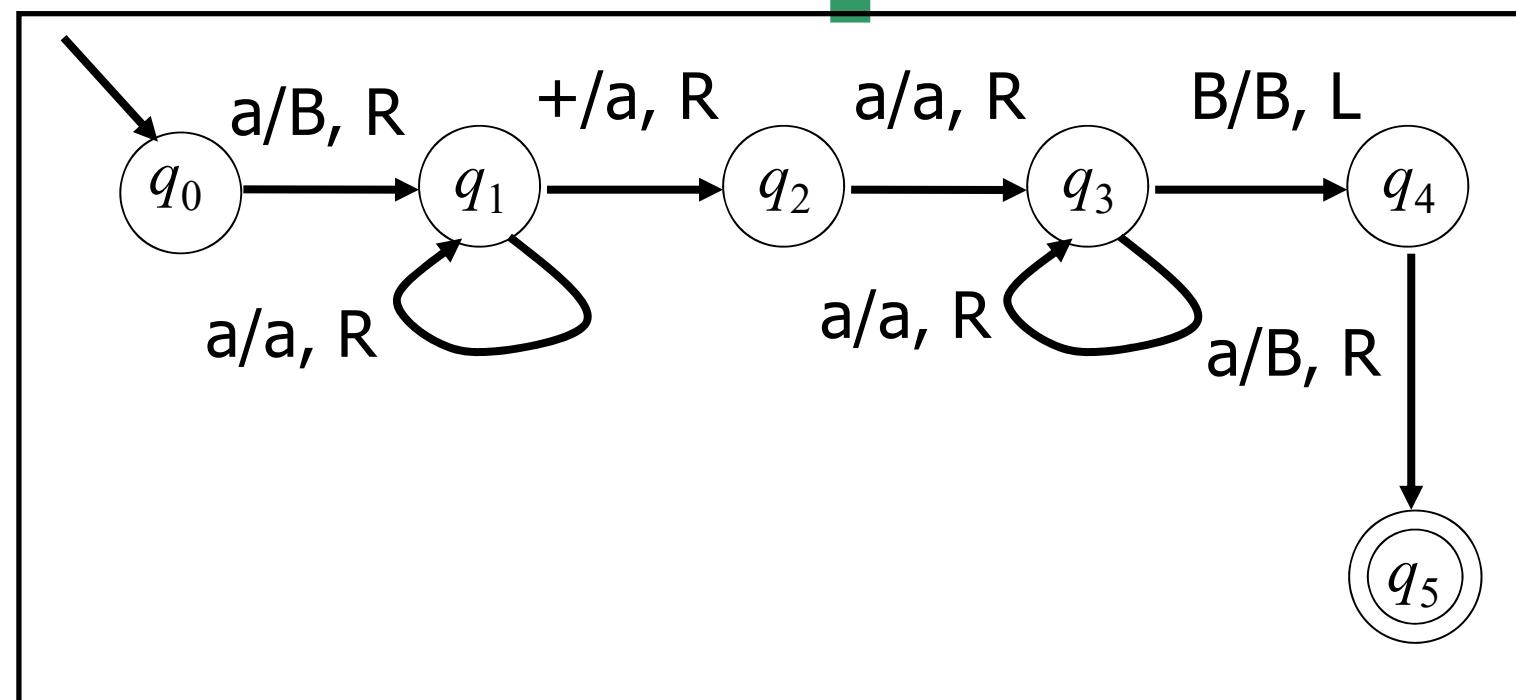
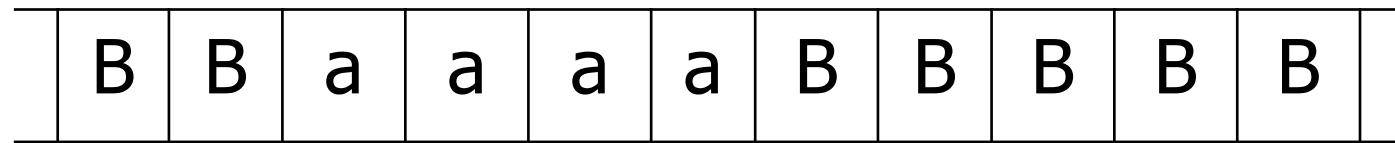
1進表現による加算のためのTM(8)



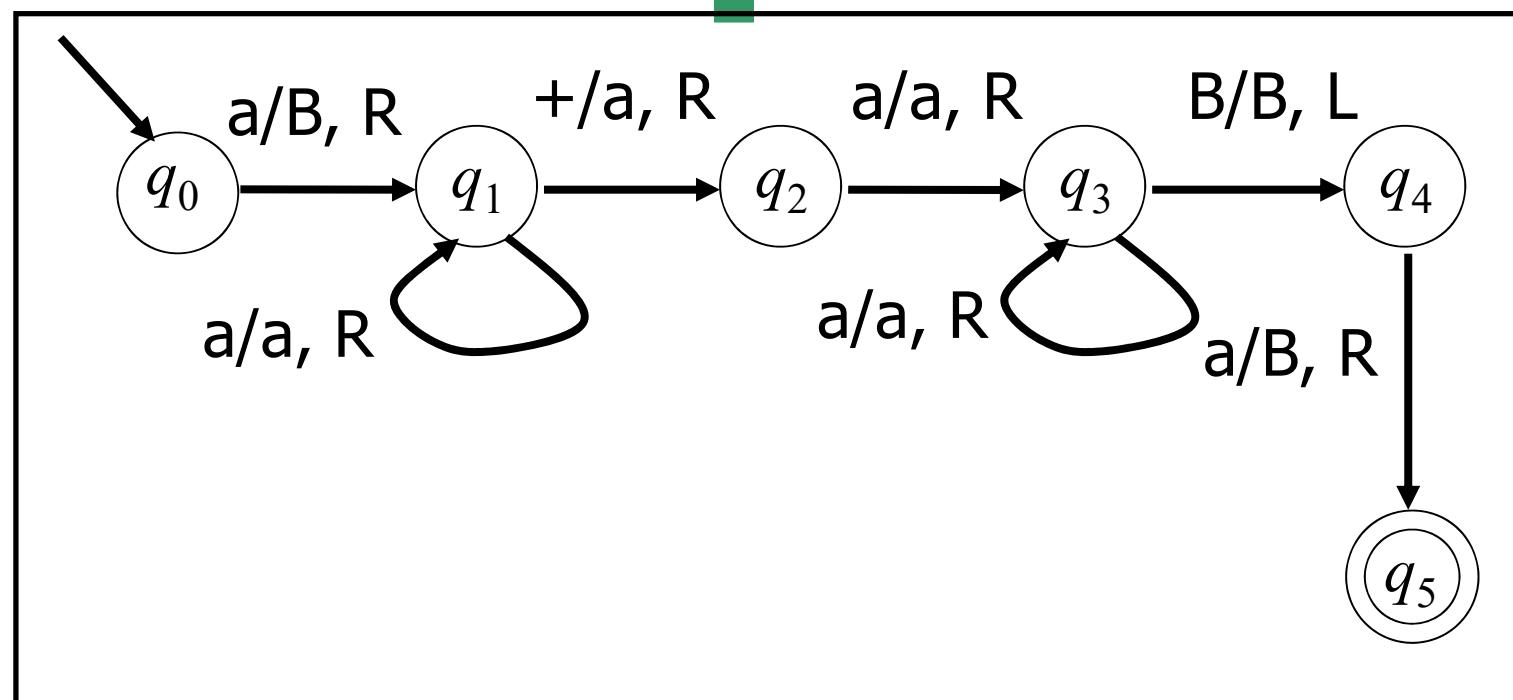
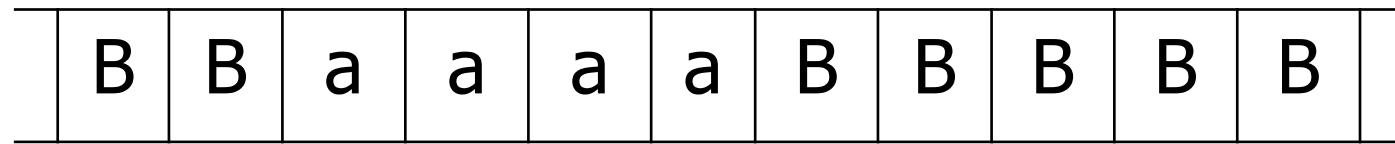
1進表現による加算のためのTM(9)



1進表現による加算のためのTM(10)

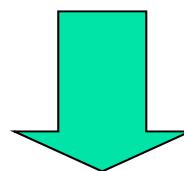


1進表現による加算のためのTM(11)



“計算”の定義

ヘッドがデータの左端にあり、有限状態機械の状態が初期状態 q_0



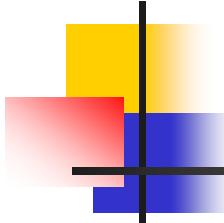
有限回の計算状況の変化

有限状態機械の状態が停止状態

計算

“計算”ではない場合

- 有限回の変化の後、有限状態機械が停止状態ではないが、計算状況が変化させられない
- 状況変化が無限に続き、有限状態機械が永久に停止状態に到達しない



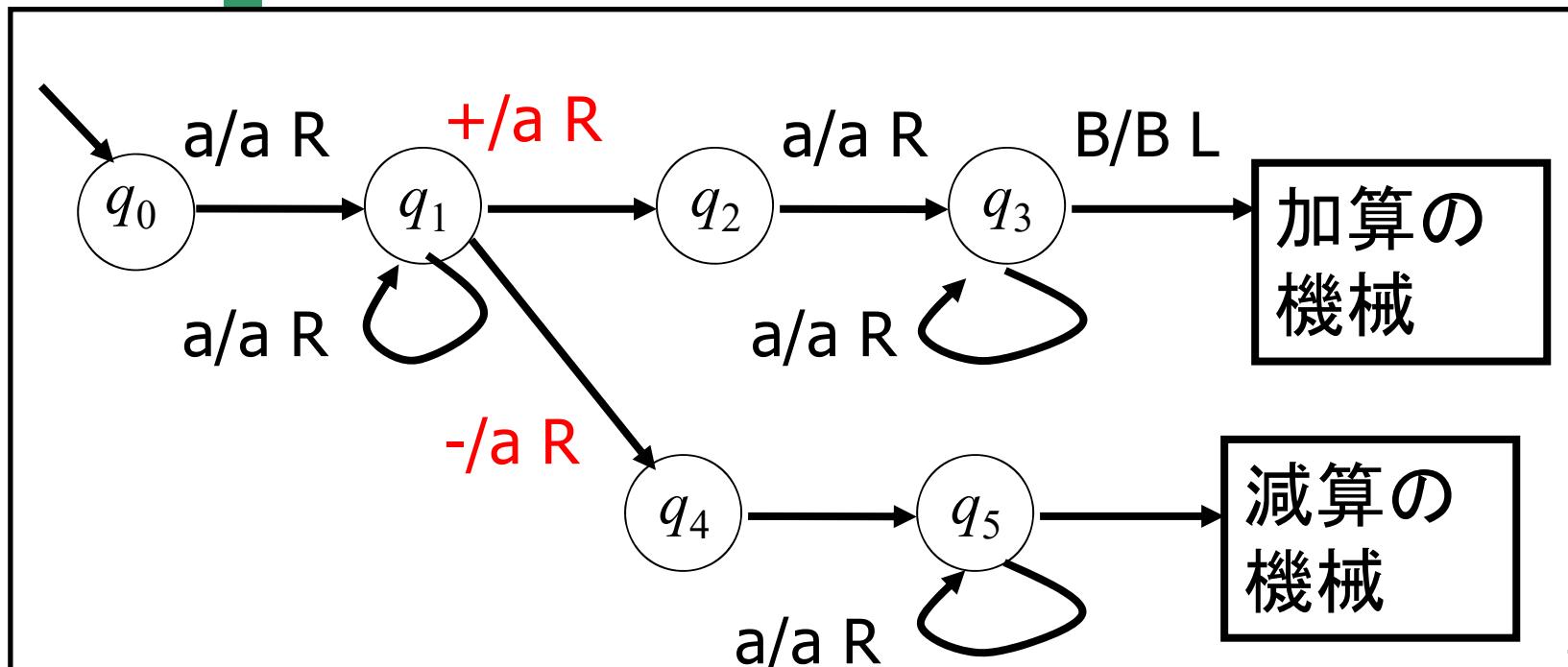
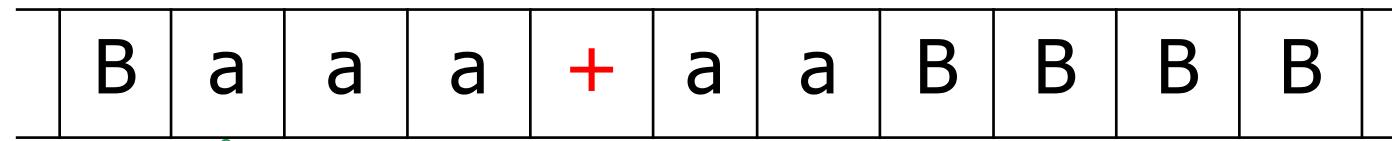
直感的な“計算”との差

- 式として正しいかどうかをチェックして、正しければ、それを書き換えている
- 自然数の位取り記数法(10進表現, 2進表現など)を使っていない
- 式を見ながら答えを書いているわけではない
 - 式で指定された(和を求める場合は +)操作を読み取ることで計算を開始しているわけではない

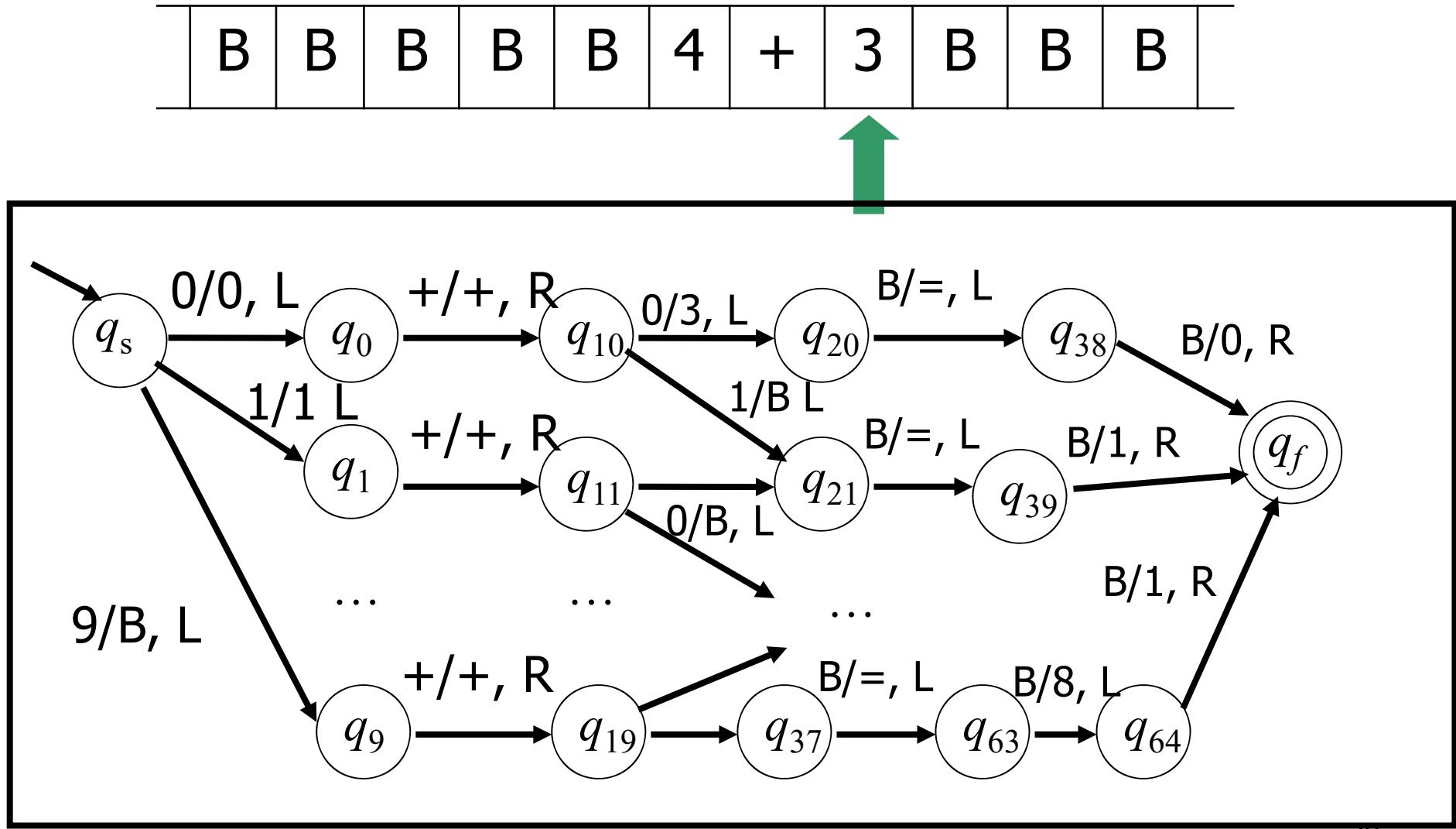
など

操作の切り替え

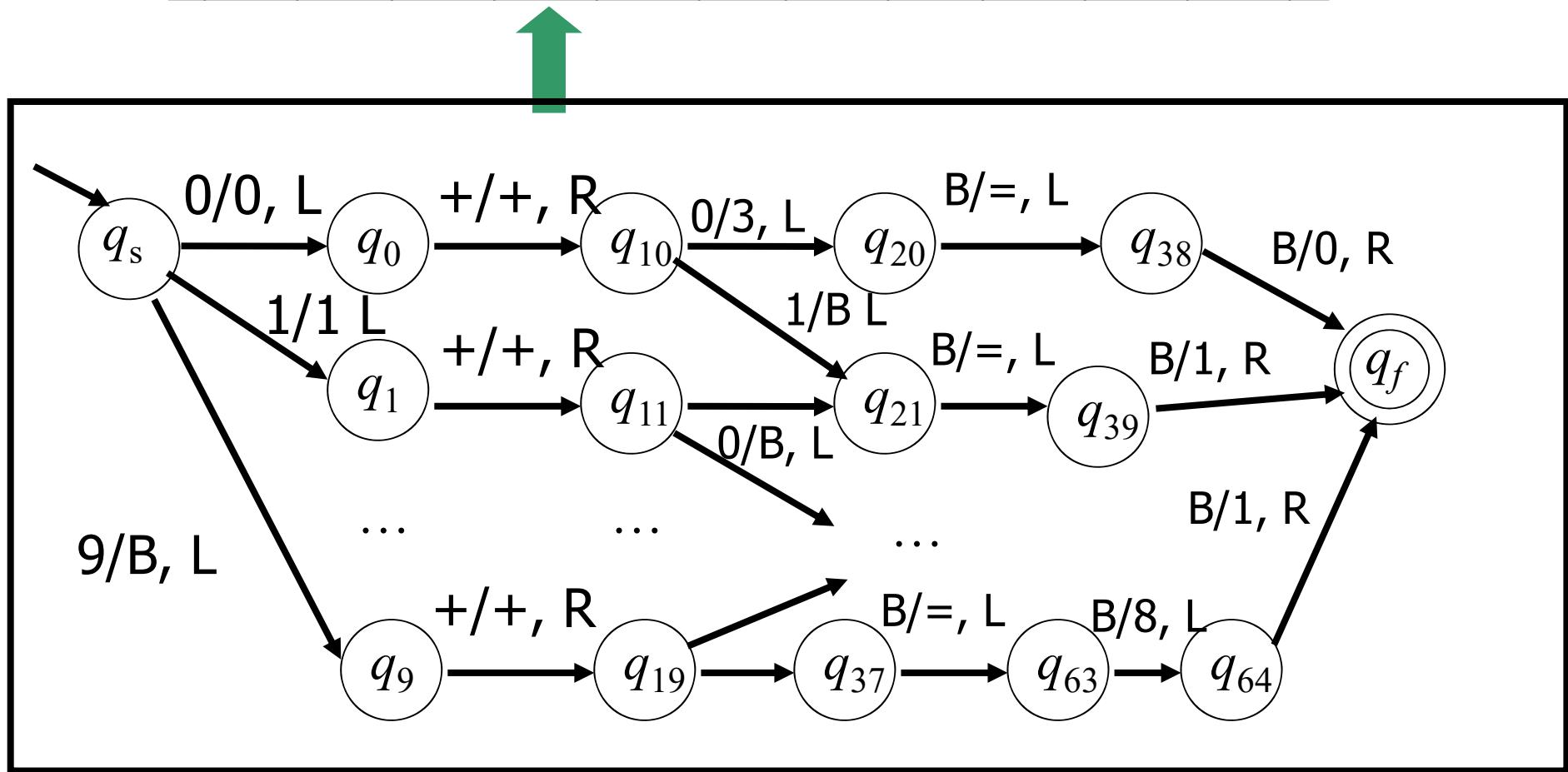
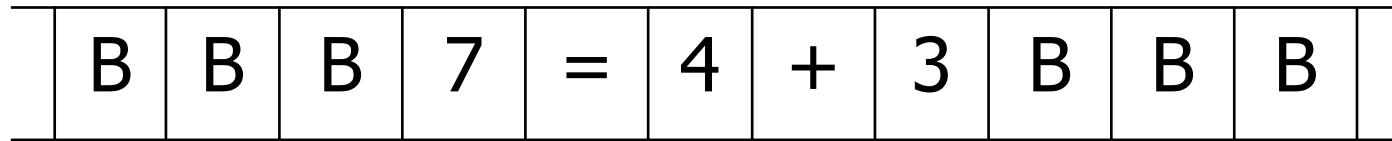
- 使われている記号が+と-いずれであるかを確認して、操作を切り替えることができる。



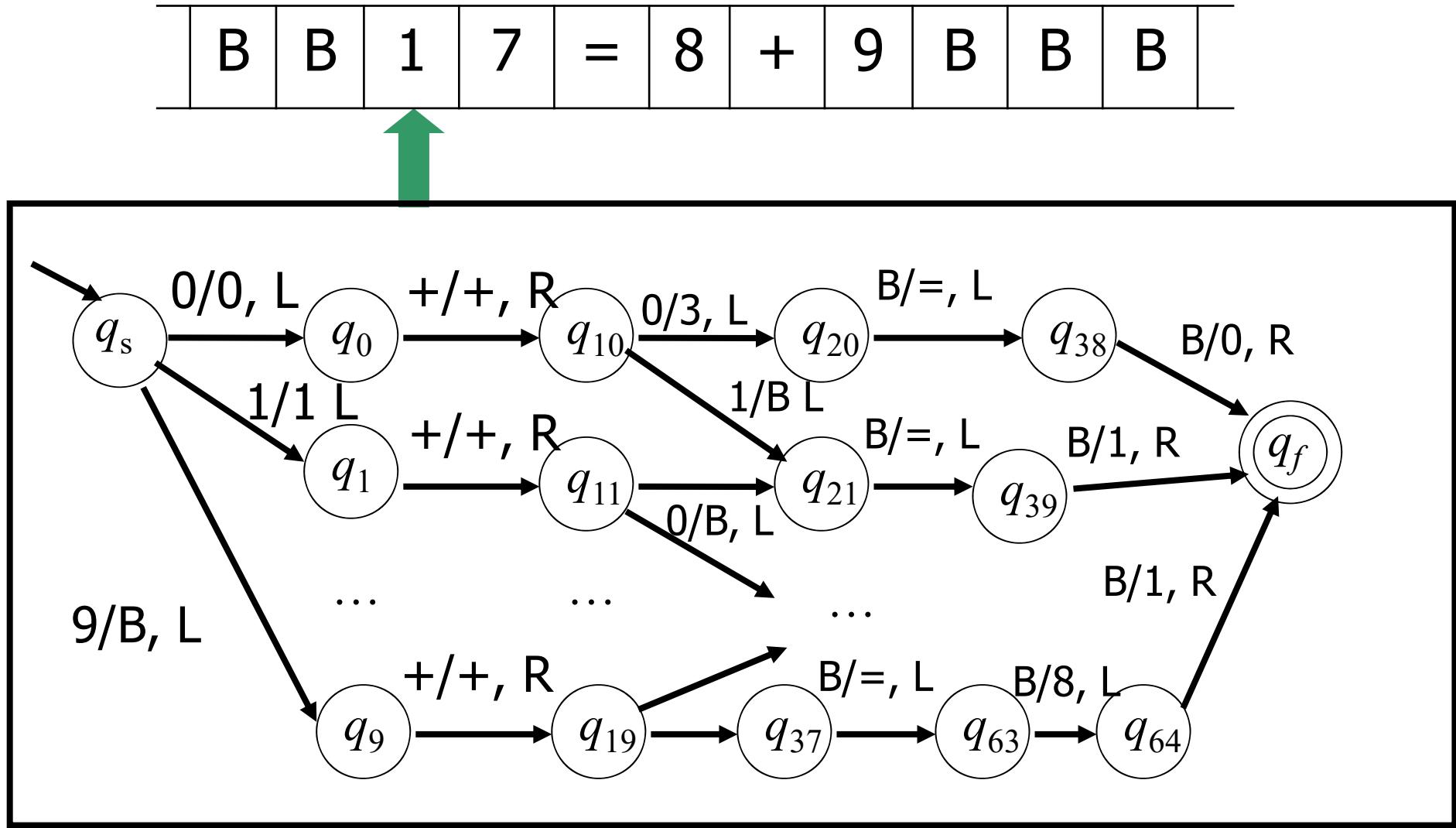
10進表現による1桁加算のためのTM(1)

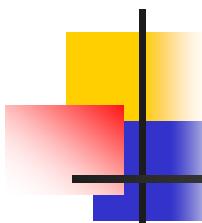


10進表現による1桁加算のためのTM



10進表現による1桁加算のためのTM



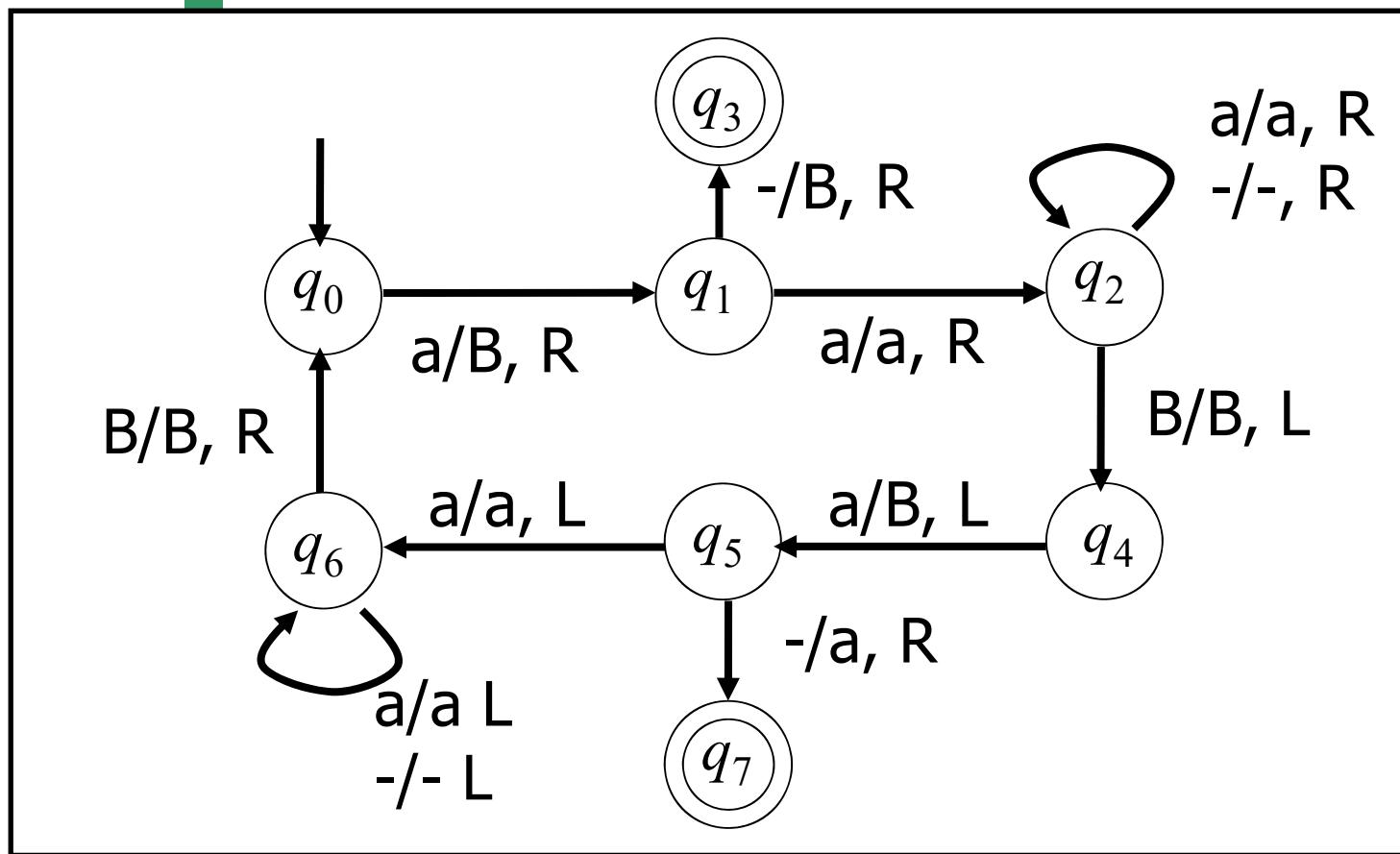
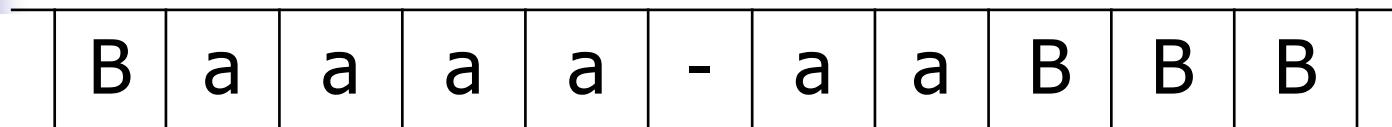


10進表現による加算のための計算

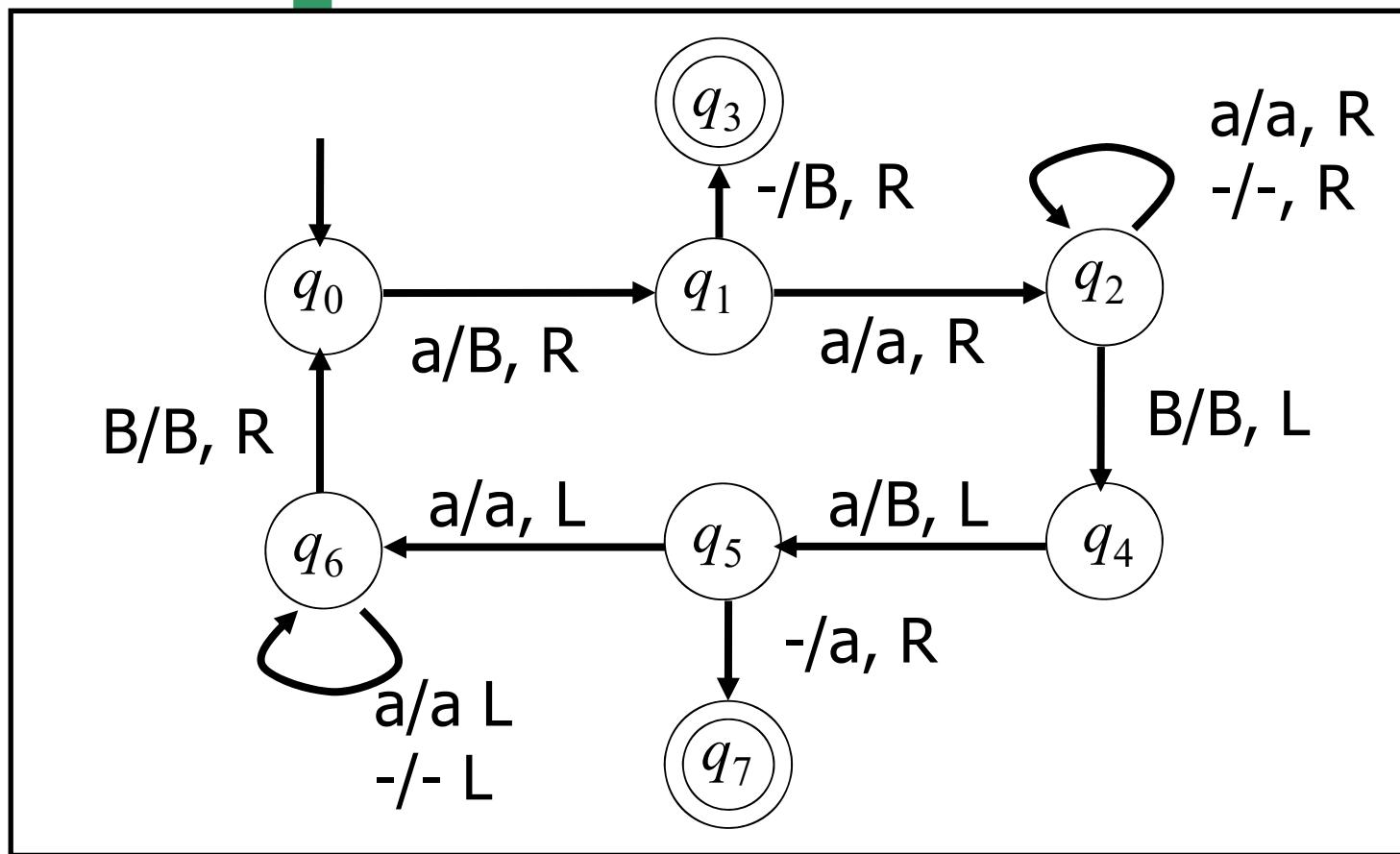
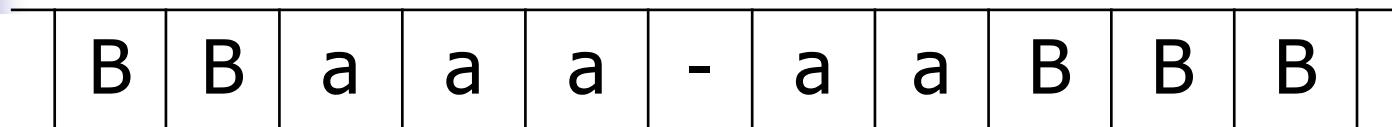
B	B	B	1	2	5	+	4	3	8	B
B	B	1	2	5	B	+	4	3	8	B
B	B	1	2	B	3	+	4	3	B	B
B	B	1	B	6	3	+	4	B	B	B
B	B	B	5	6	3	+	B	B	B	B

Diagram illustrating the step-by-step addition of two binary numbers (B) using base 10 representation. The numbers are:
Top row: B B B 1 2 5 + 4 3 8 B
Second row: B B 1 2 5 B + 4 3 8 B
Third row: B B 1 2 B 3 + 4 3 B B
Fourth row: B B 1 B 6 3 + 4 B B B
Bottom row: B B B 5 6 3 + B B B B
Green arrows point from right to left between the columns, indicating the propagation of the carry bit (B) from the result of one column to the next.

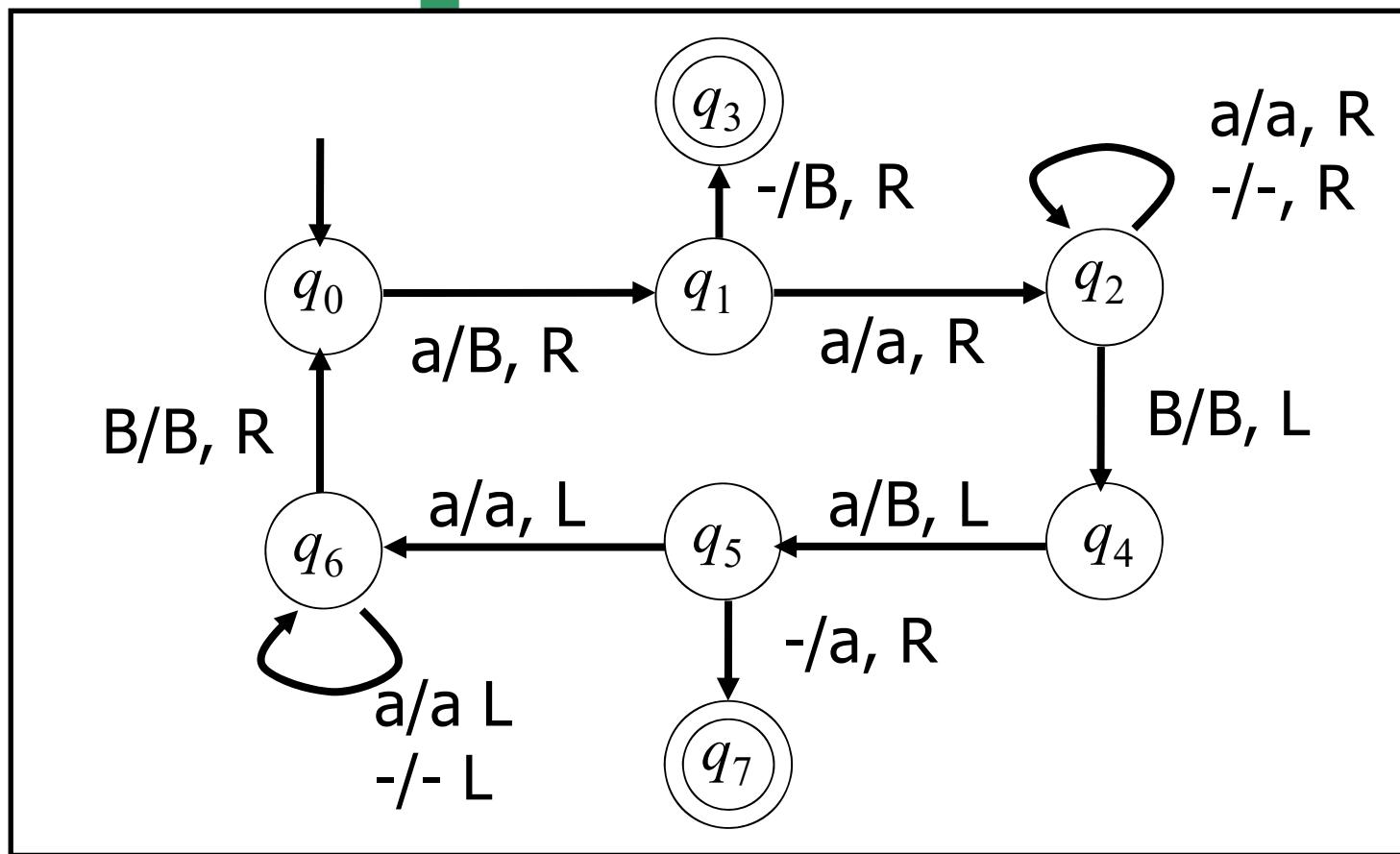
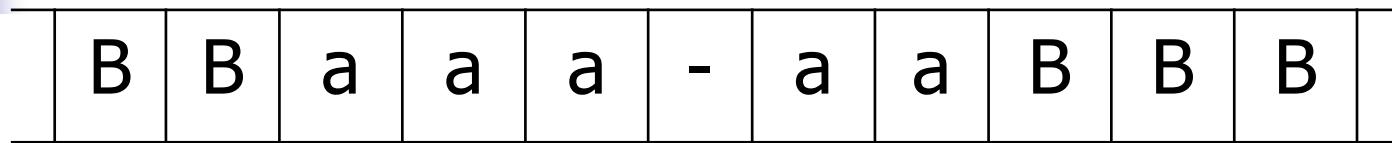
1進表現による減算|x-y|(1-1)



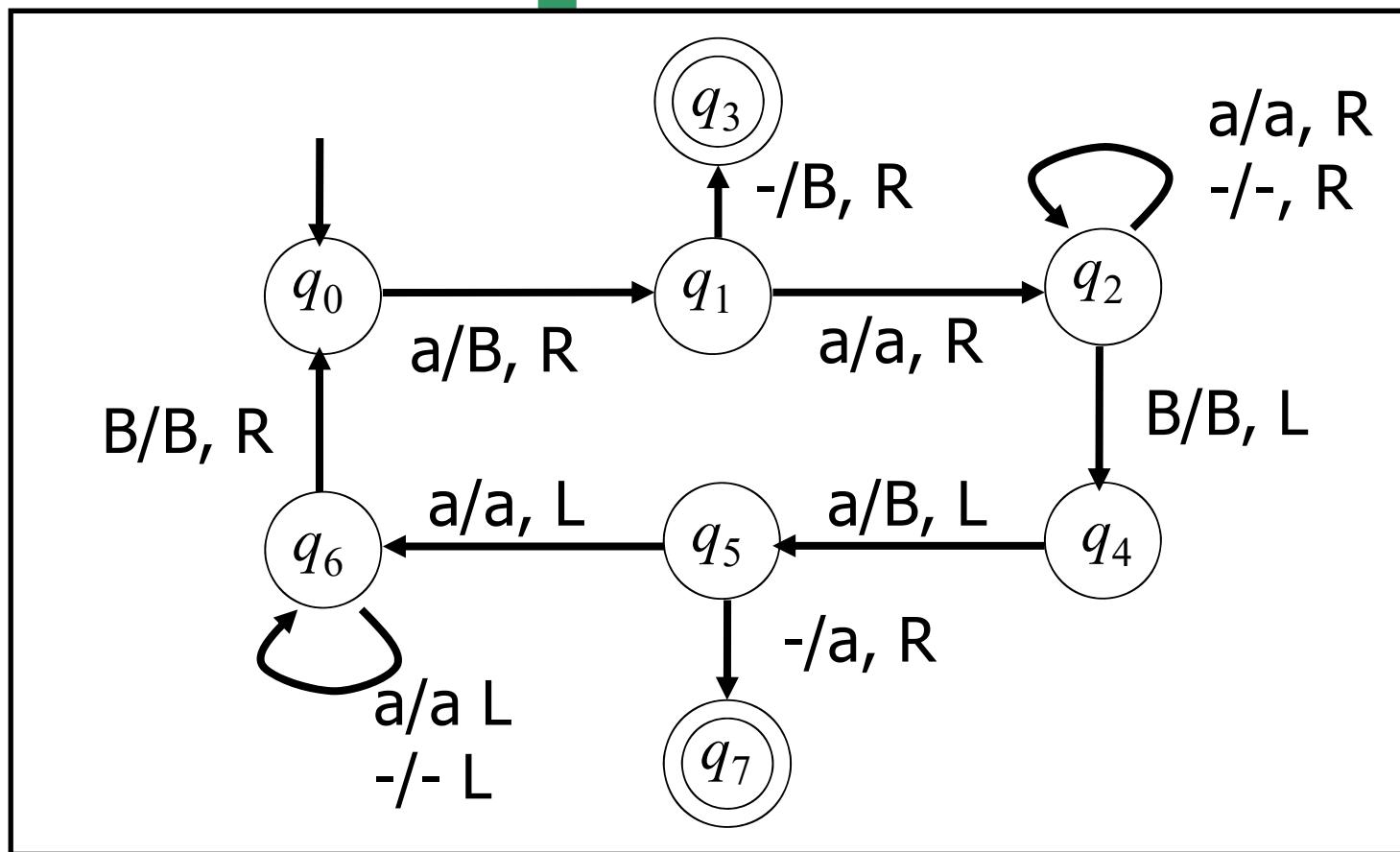
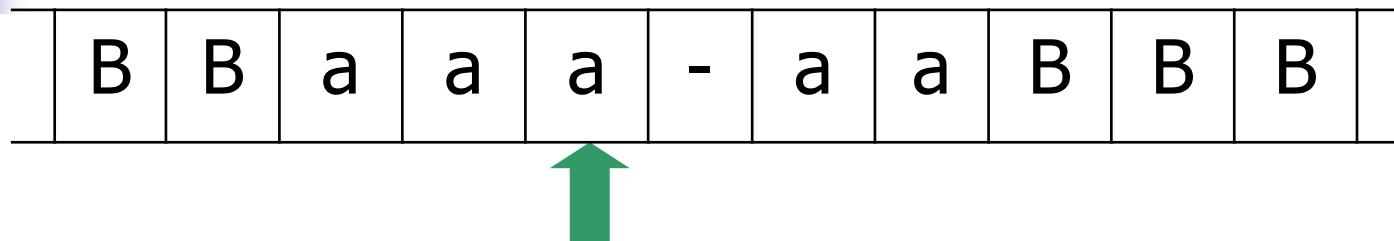
1進表現による減算|x-y|(1-2)



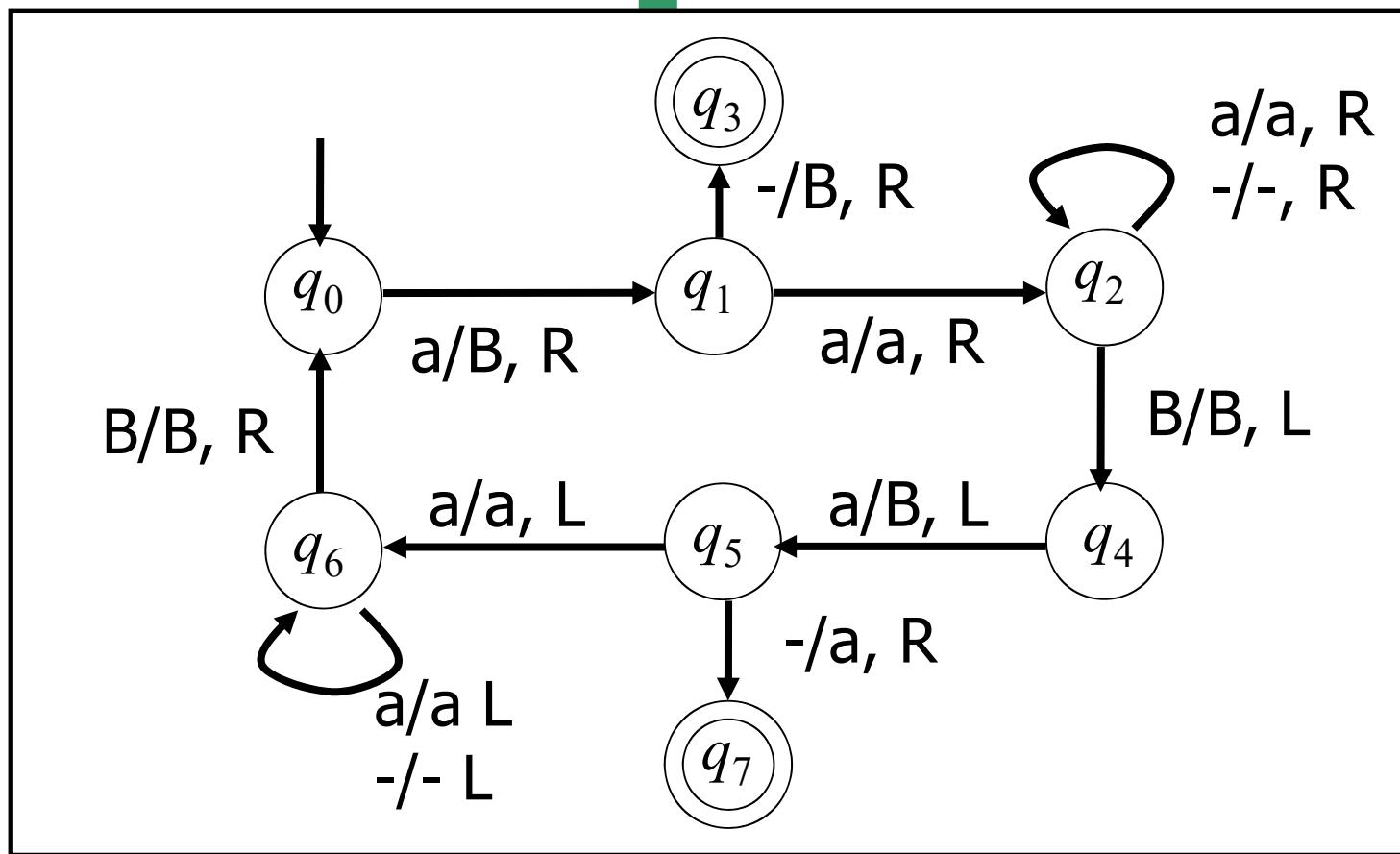
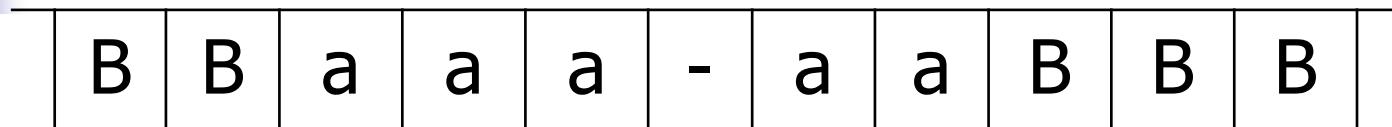
1進表現による減算 $|x-y|$ (1-3)



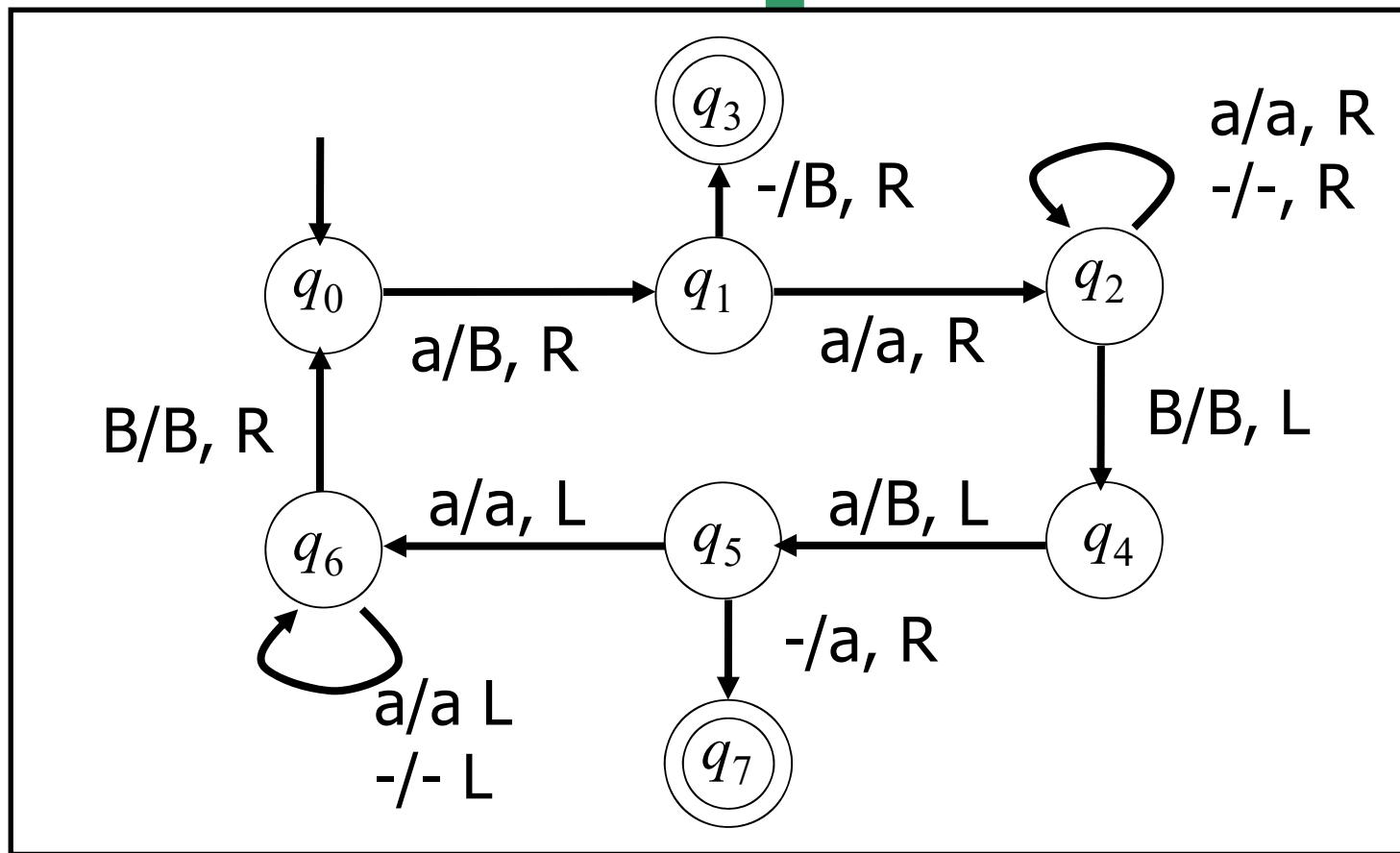
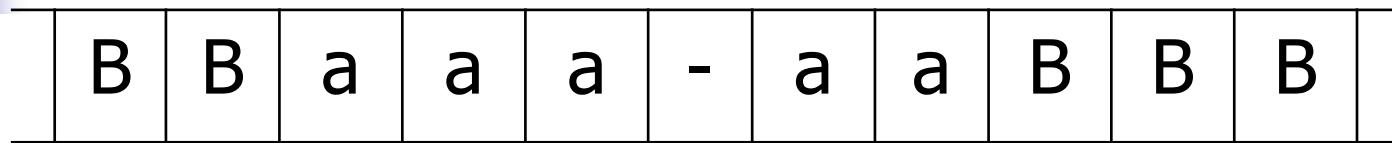
1進表現による減算 $|x-y|$ (1-4)



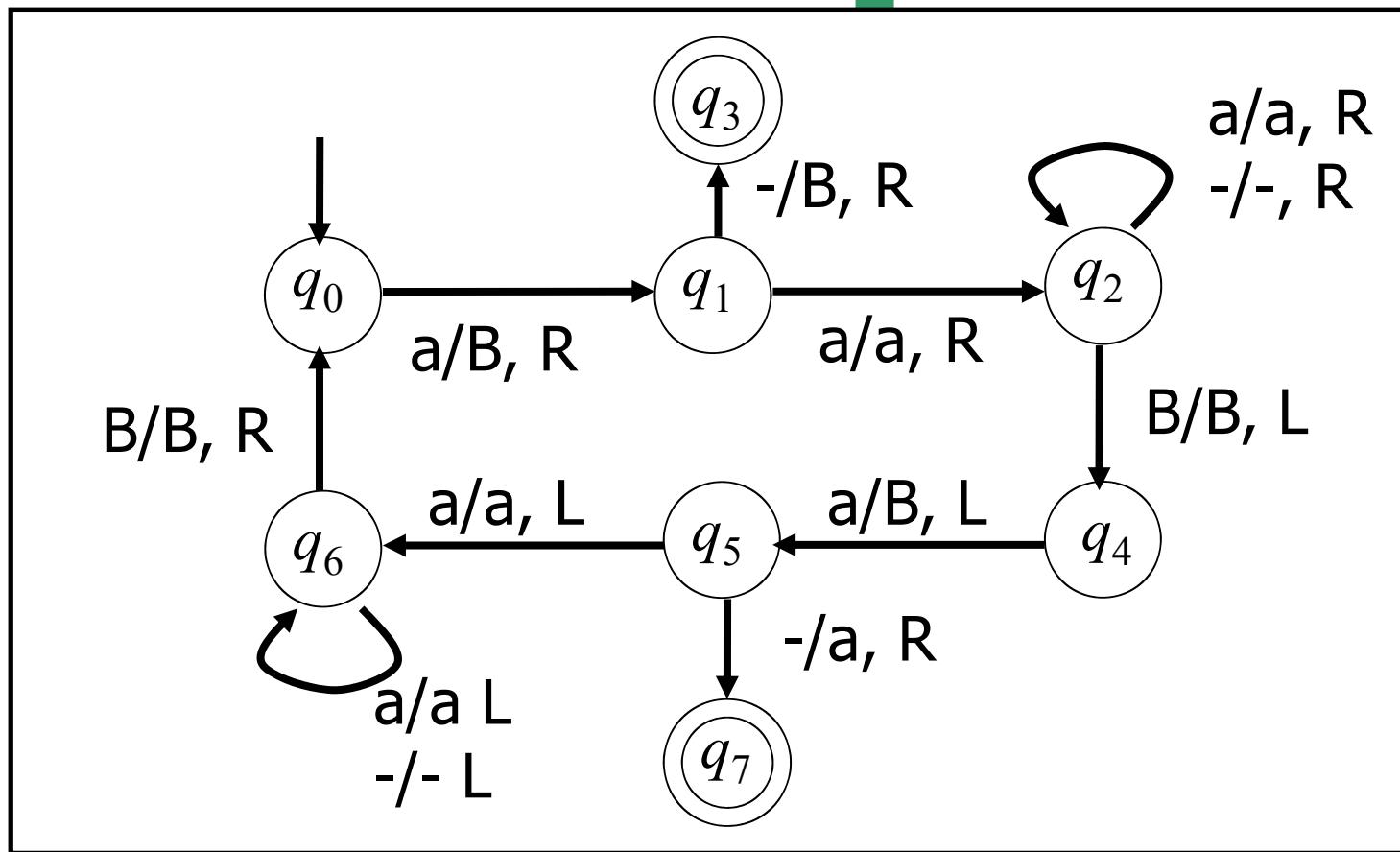
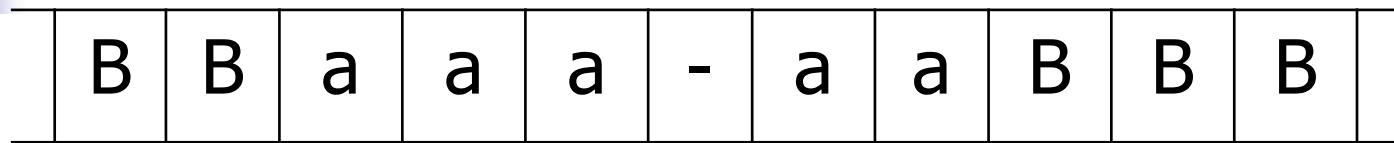
1進表現による減算|x-y|(1-5)



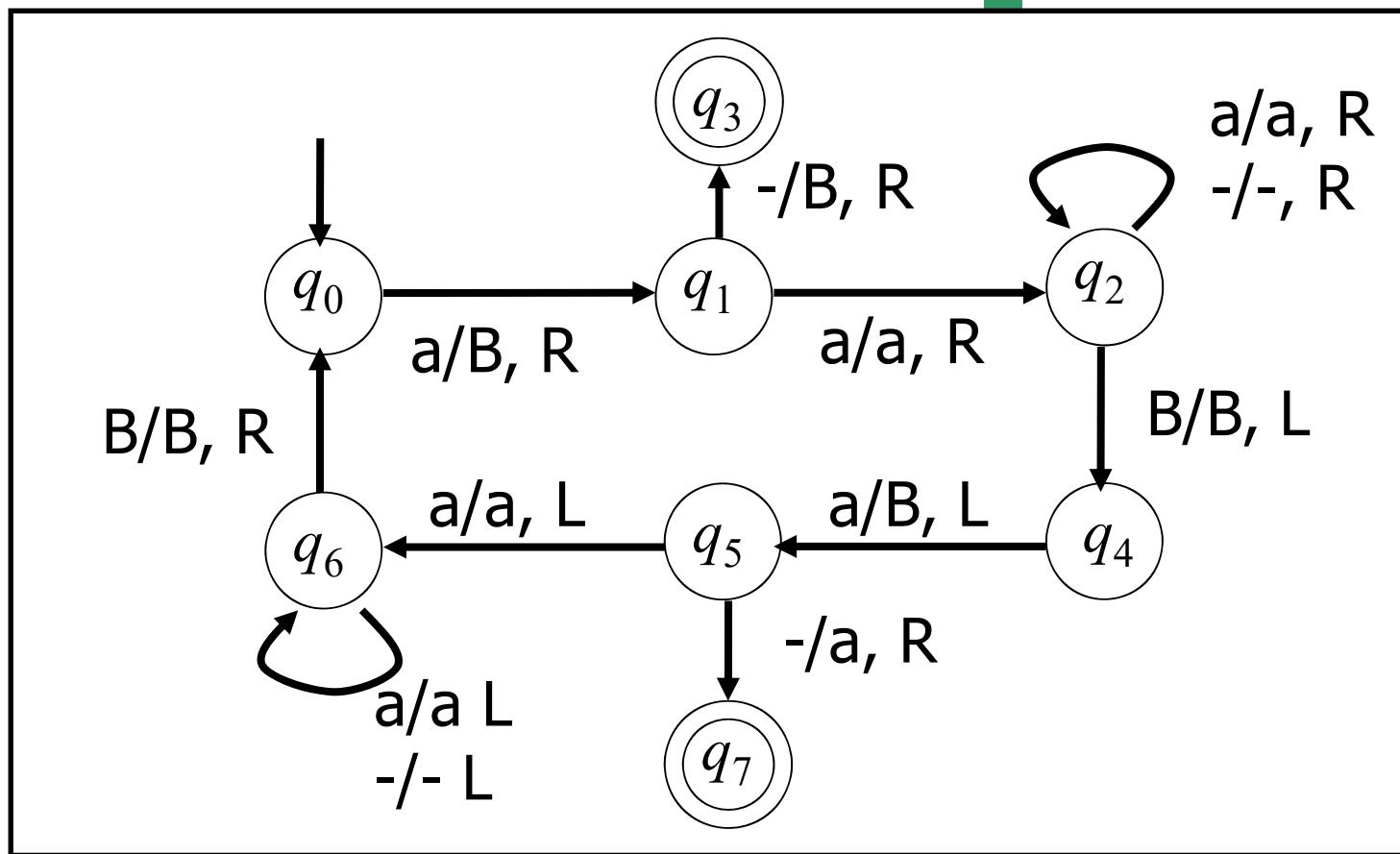
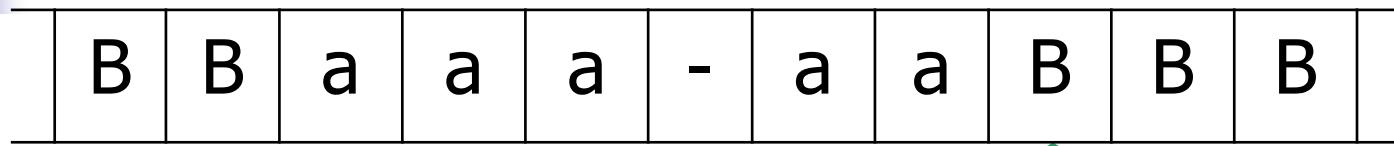
1進表現による減算 $|x-y|$ (1-6)



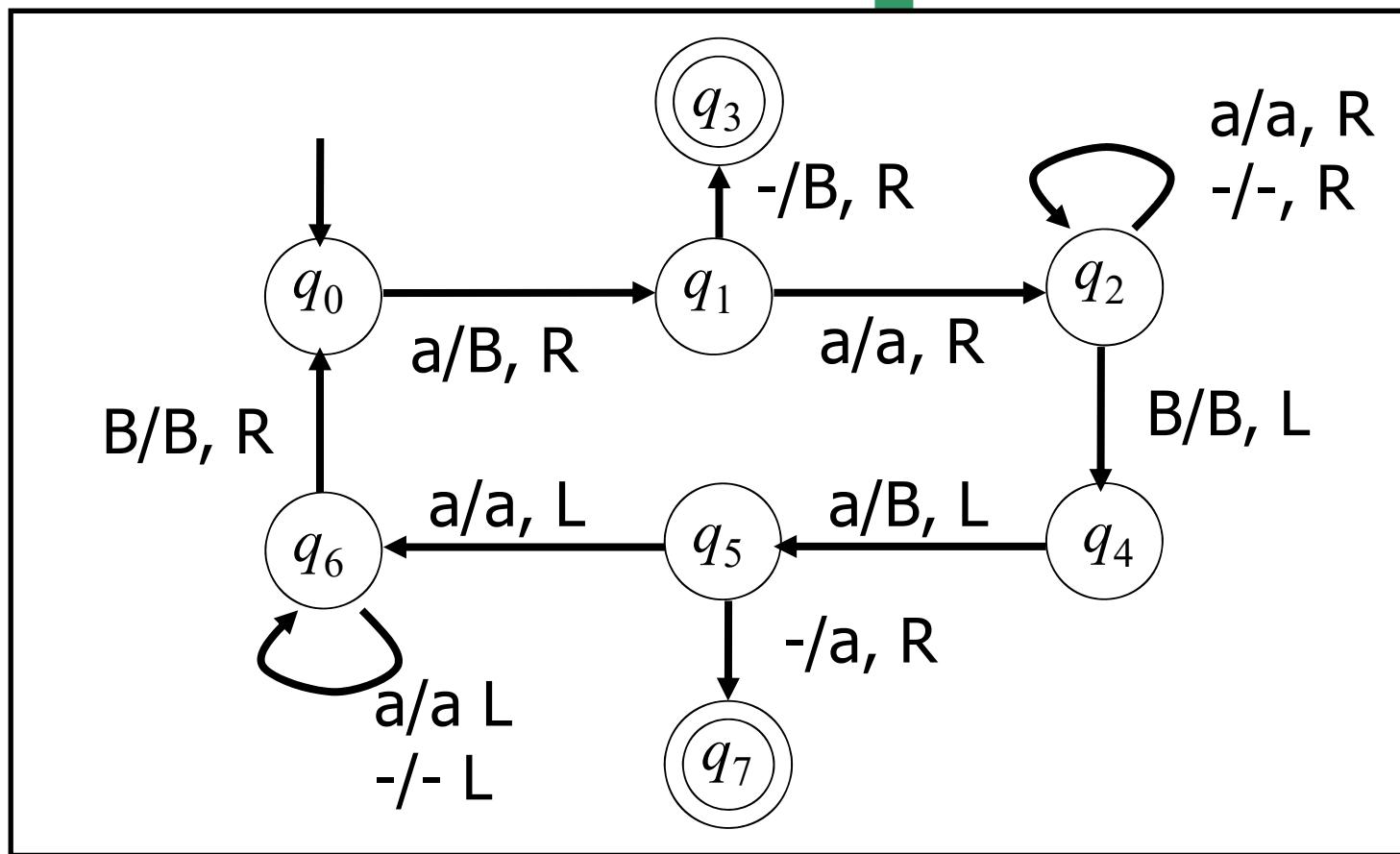
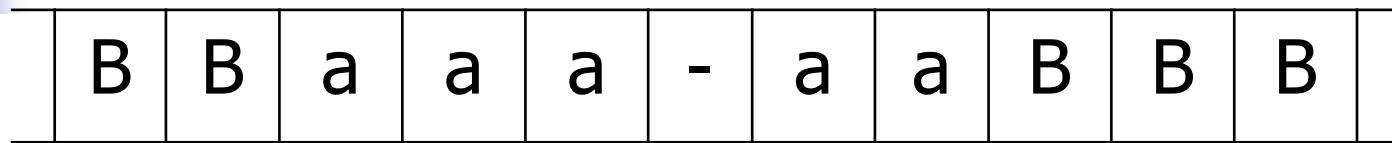
1進表現による減算 $|x-y|$ (1-7)



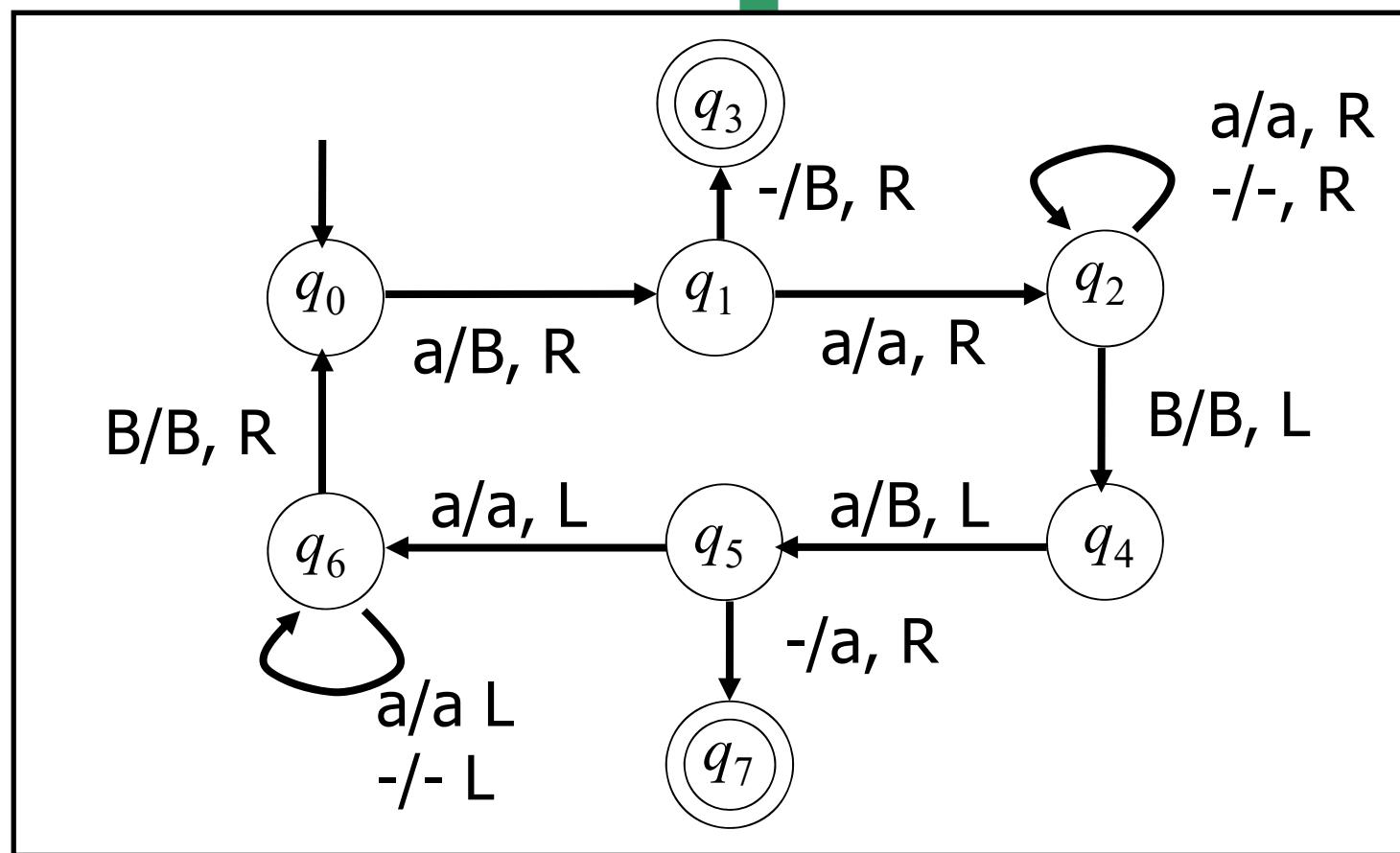
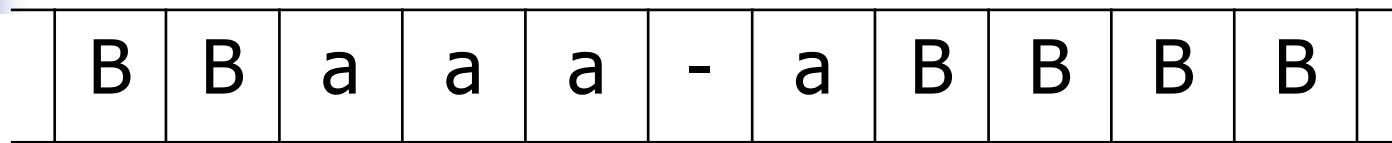
1進表現による減算 $|x-y|$ (1-8)



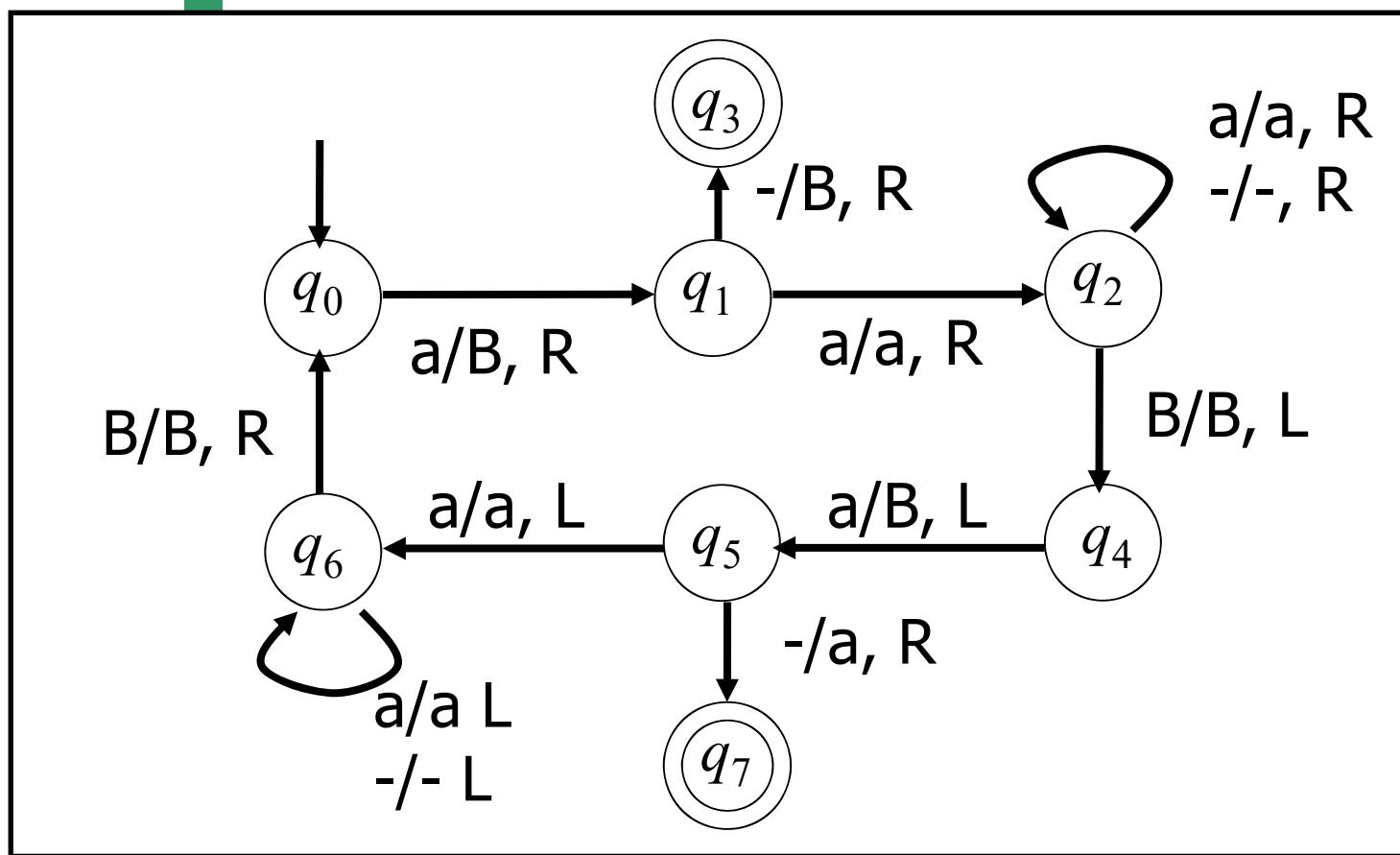
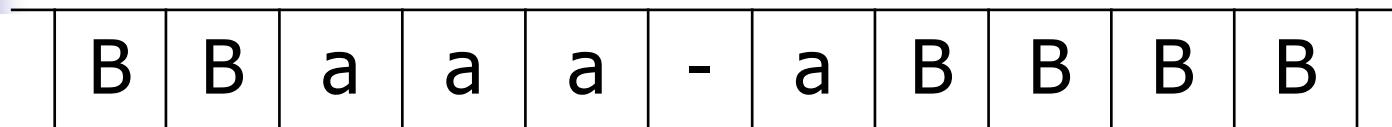
1進表現による減算 $|x-y|$ (1-9)



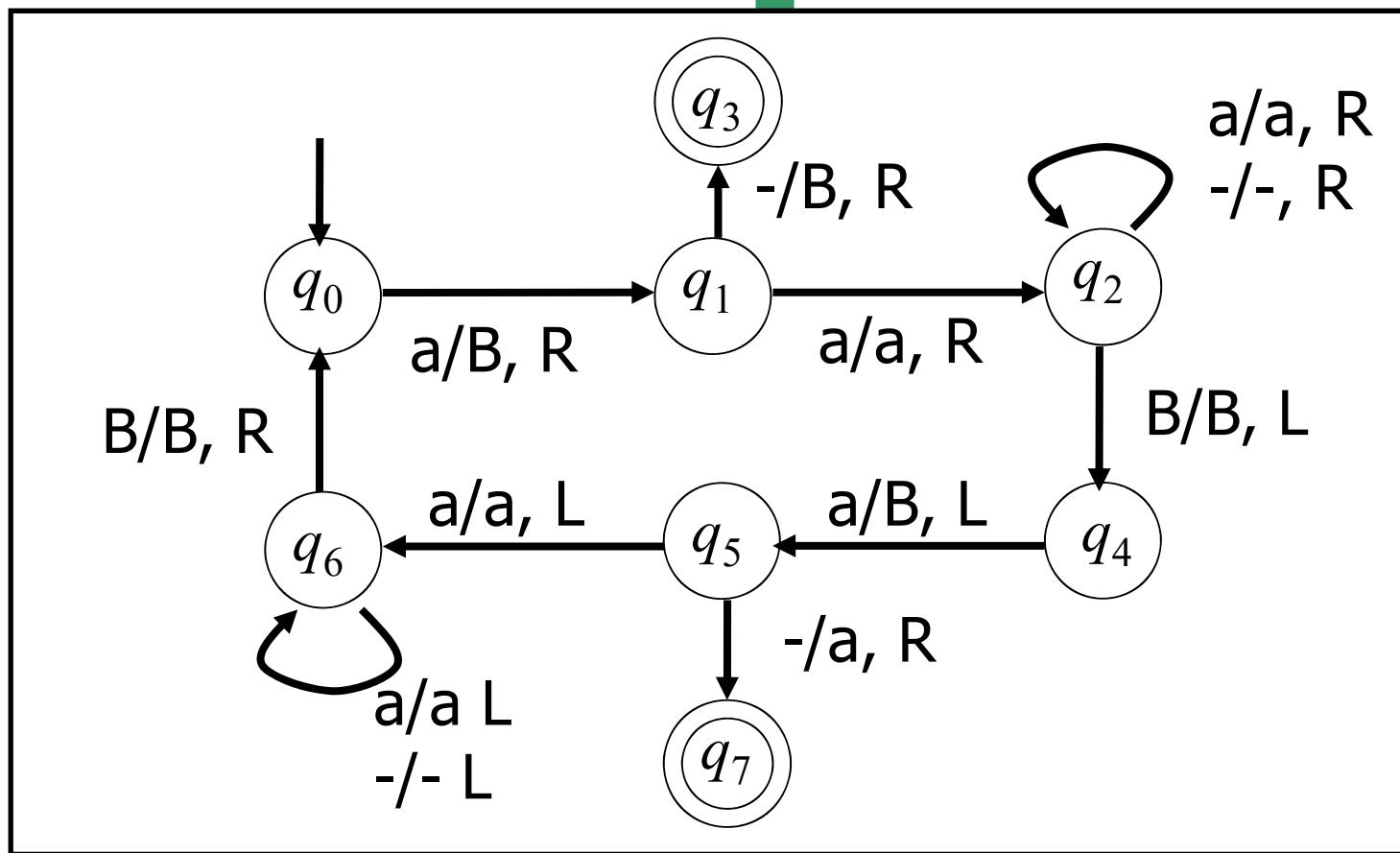
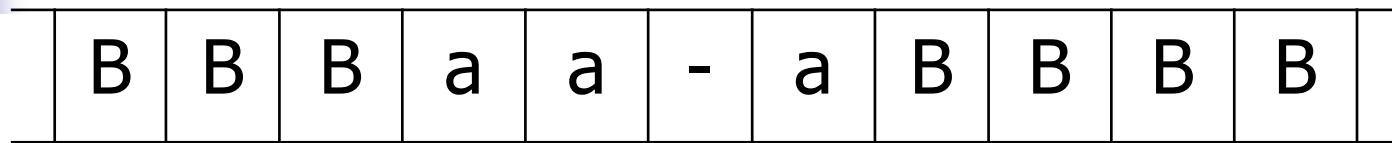
1進表現による減算 $|x-y|$ (1-10)



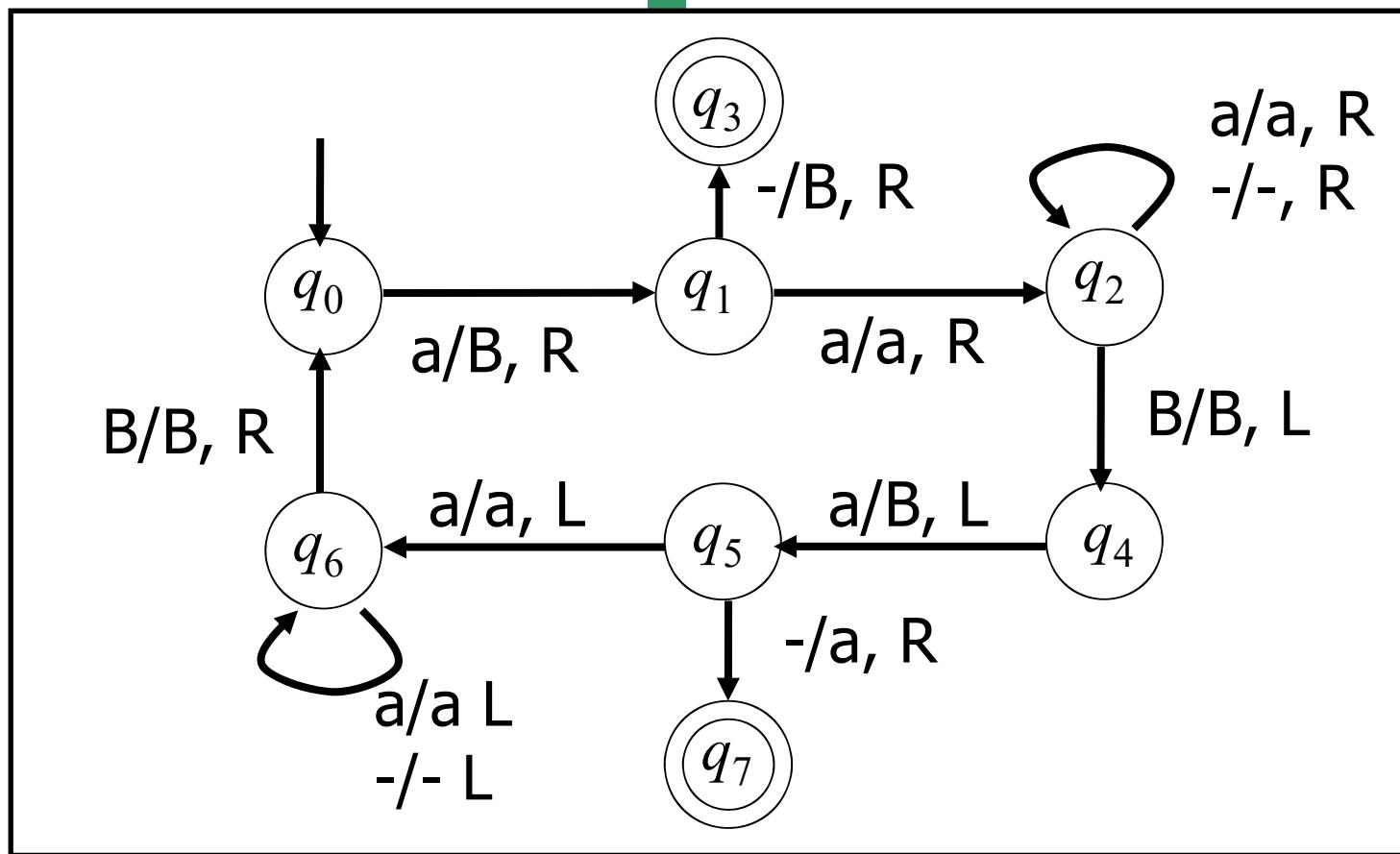
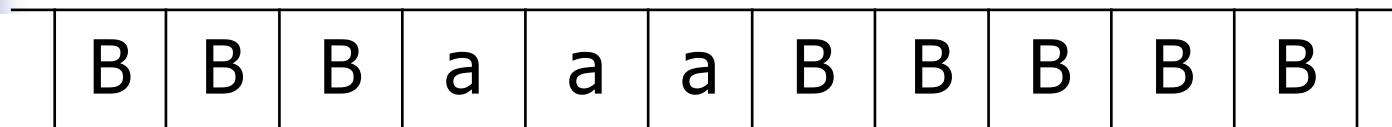
1進表現による減算|x-y|(1-15)



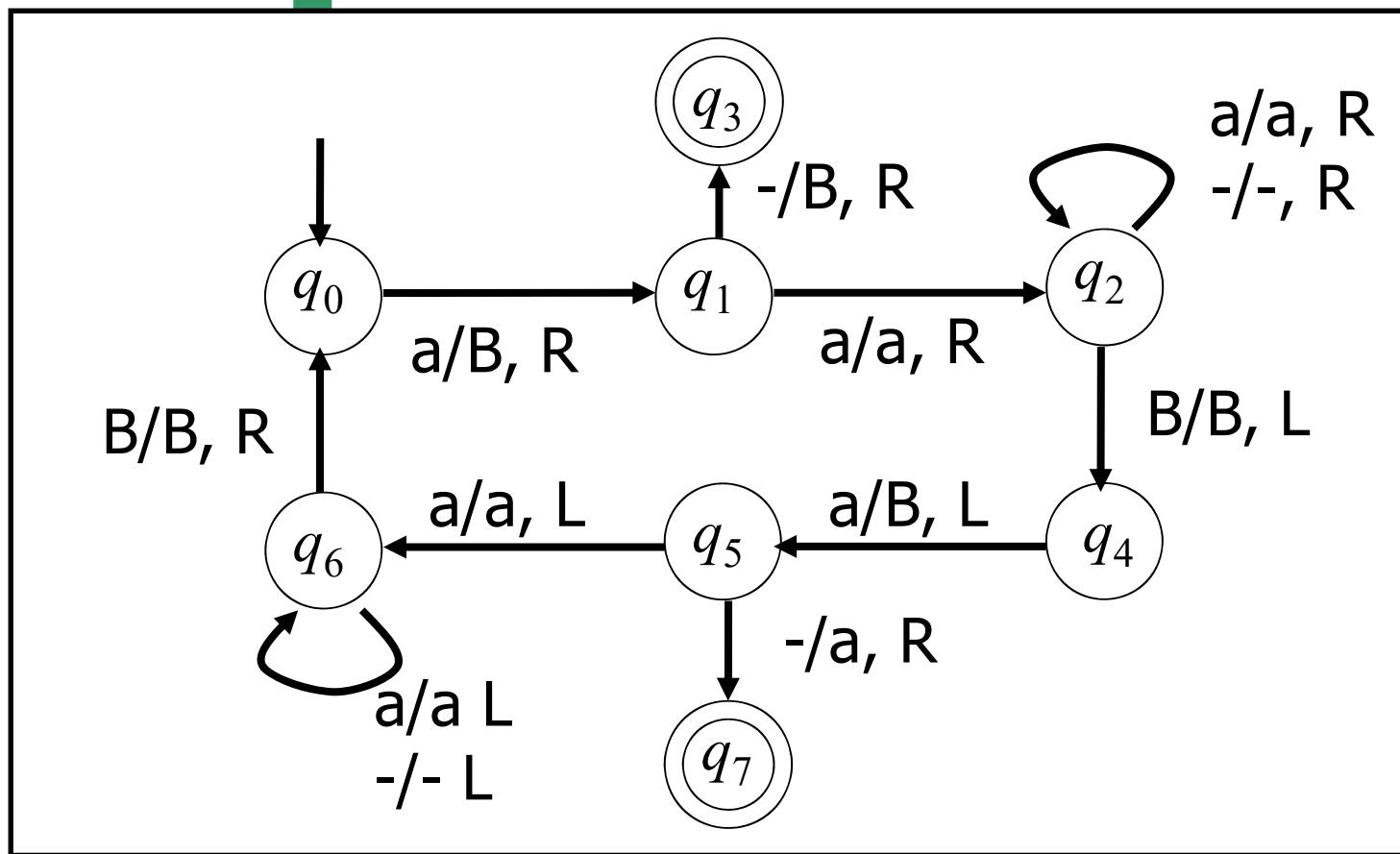
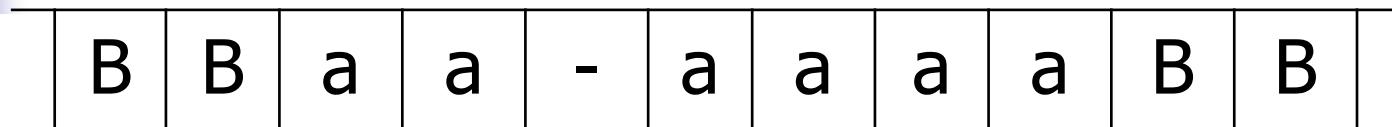
1進表現による減算 $|x-y|$ (1-22)



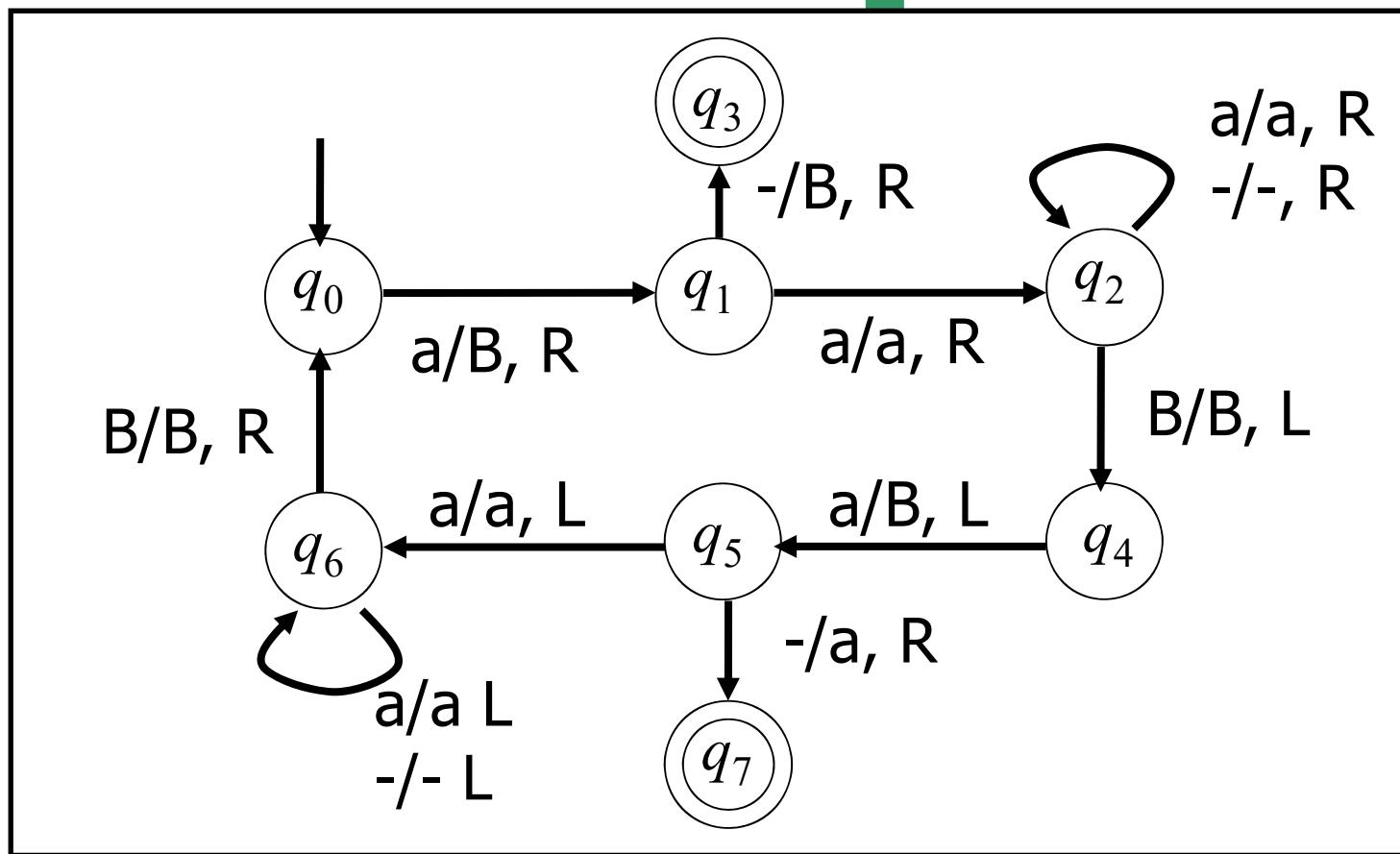
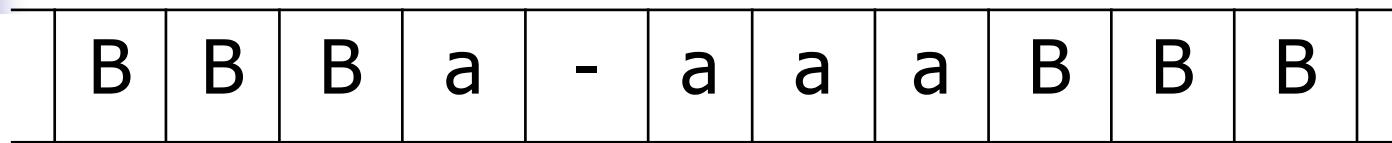
1進表現による減算|x-y|(1-23)



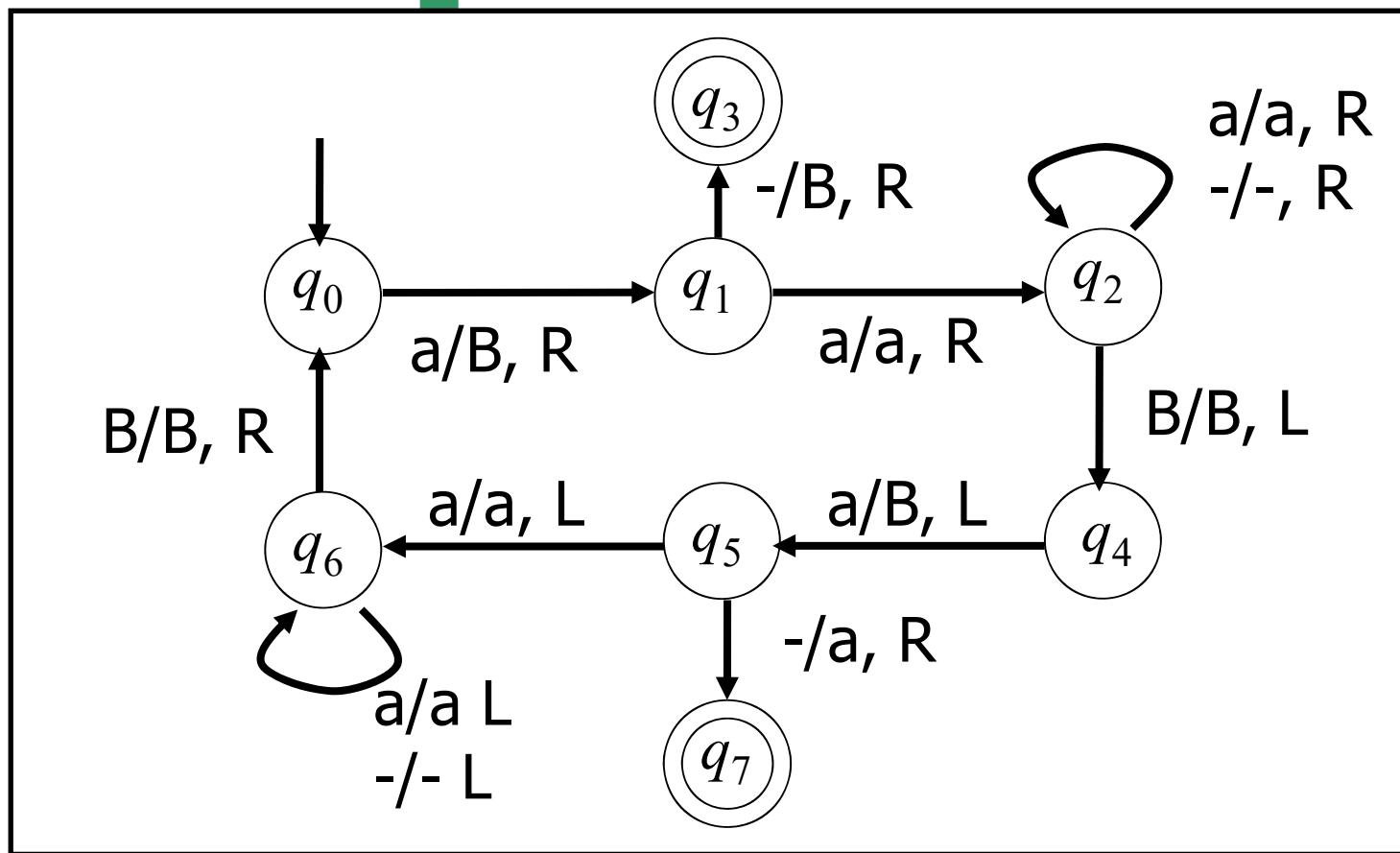
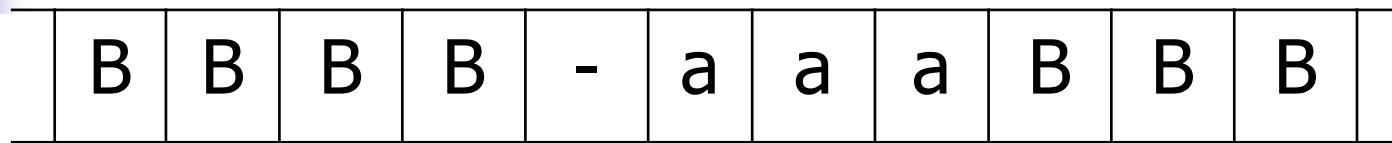
1進表現による減算 $|x-y|$ (2-1)



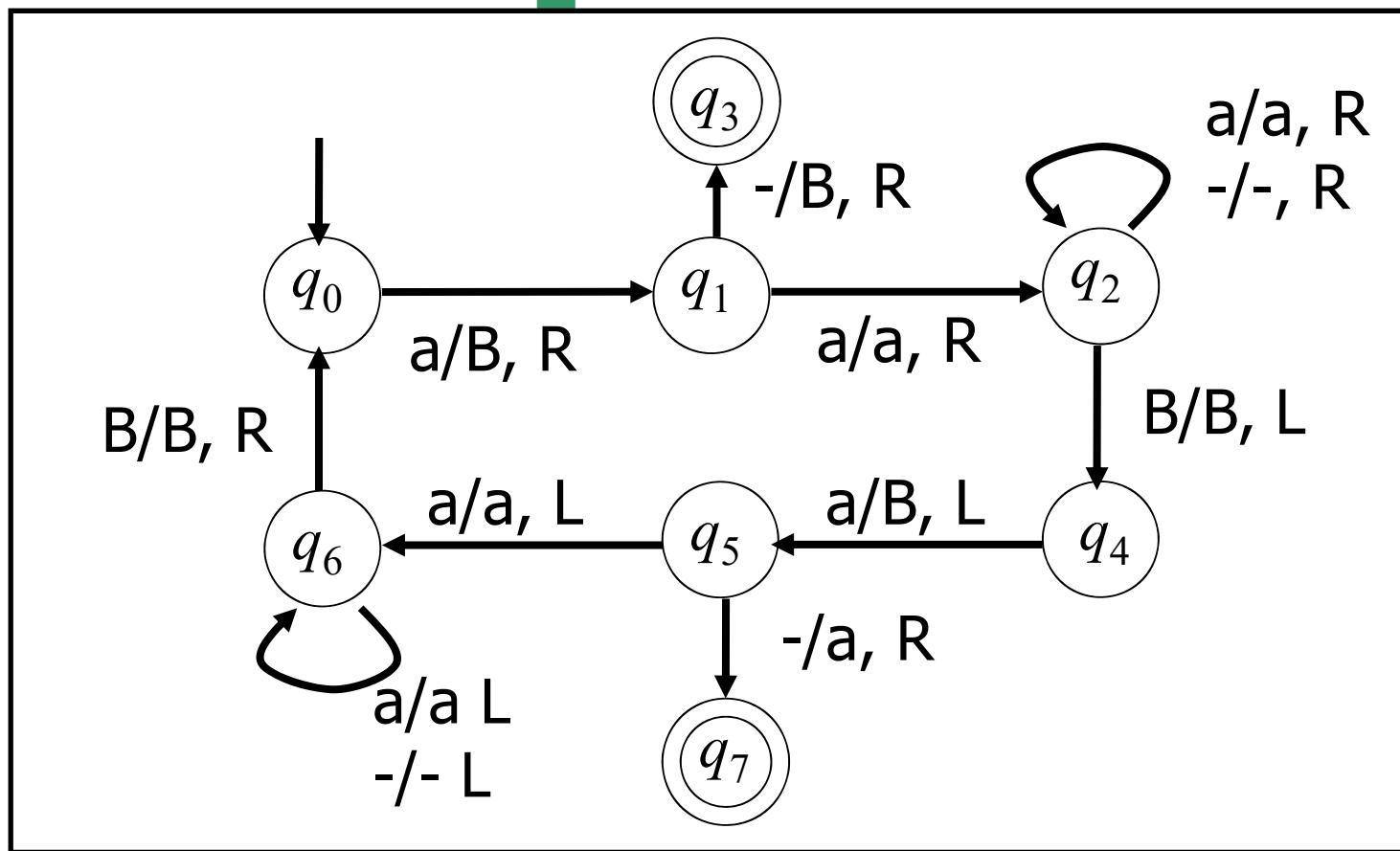
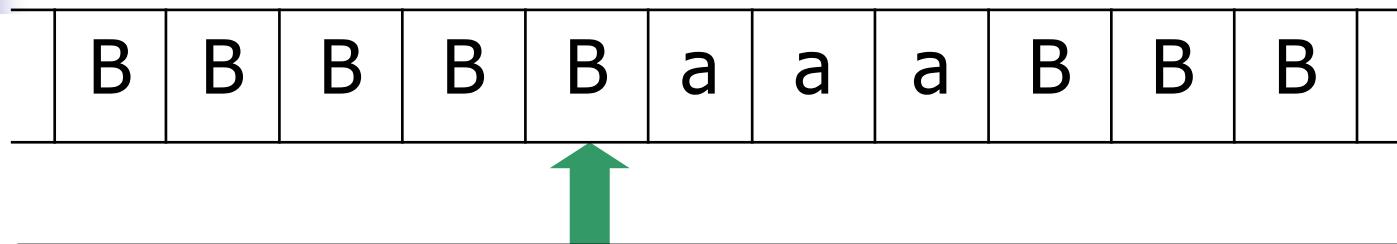
1進表現による減算 $|x-y|$ (2-8)



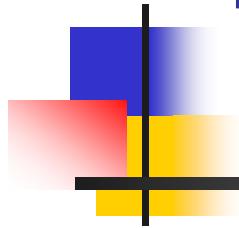
1進表現による減算|x-y|(2-14)



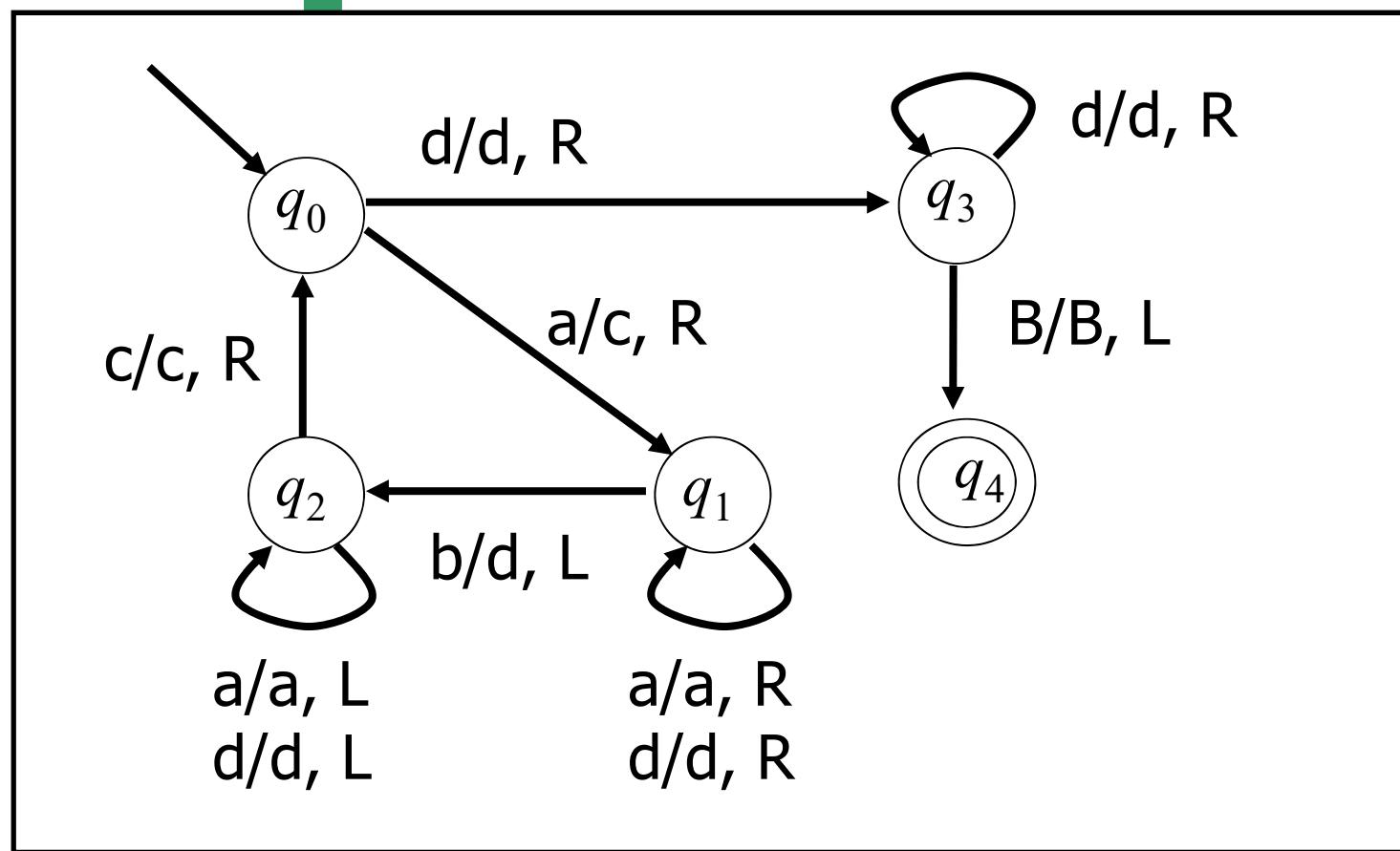
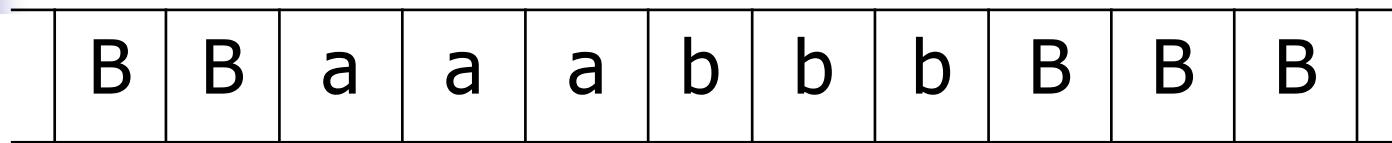
1進表現による減算|x-y|(2-15)



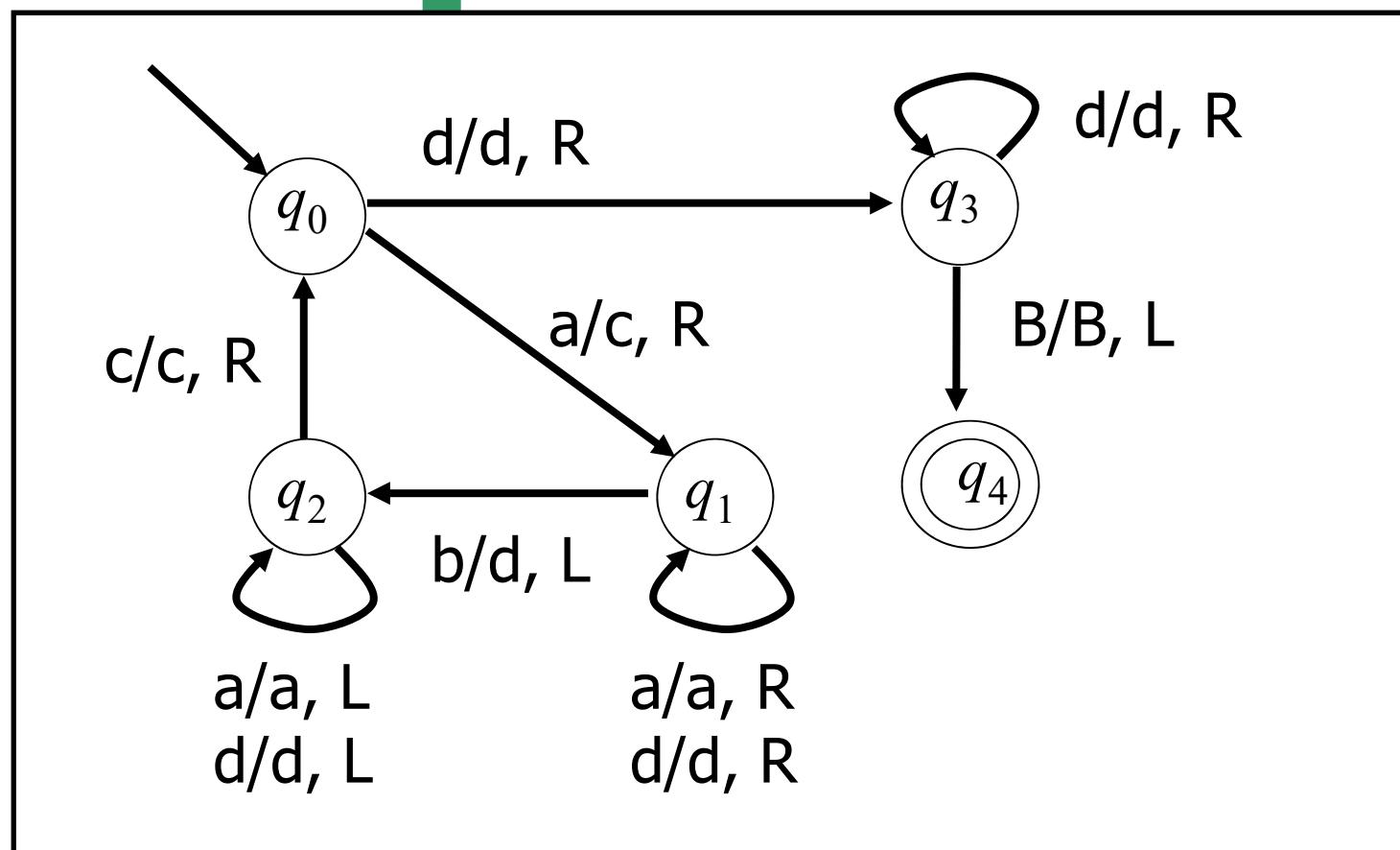
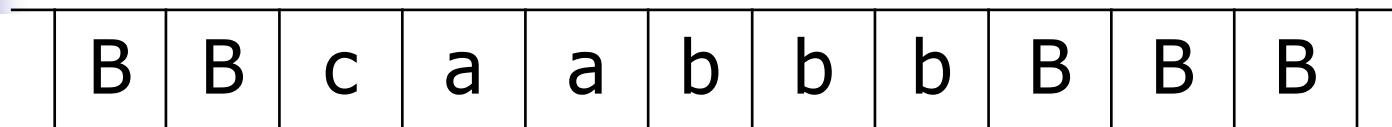
形式言語の受理機械としての Turing機械



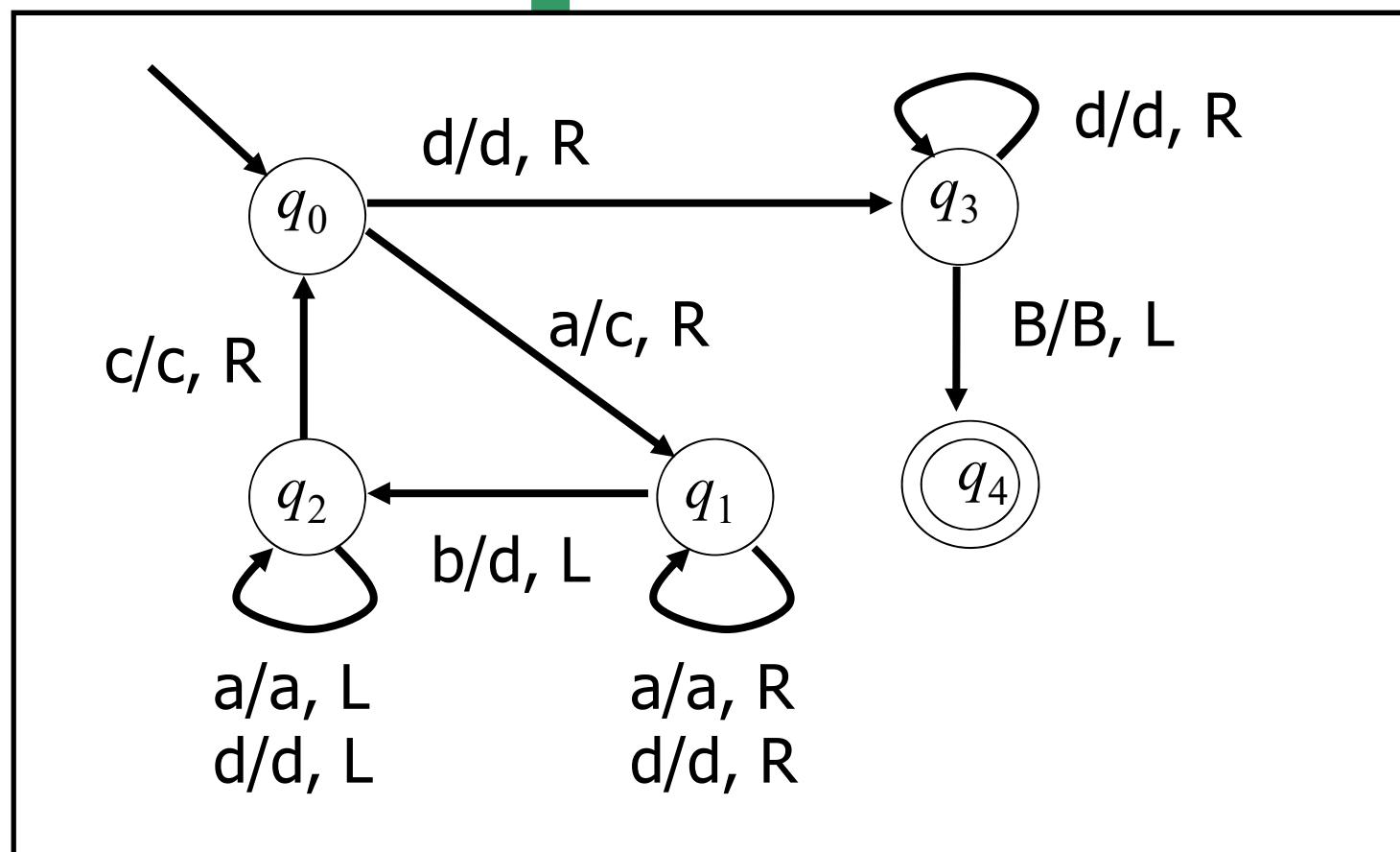
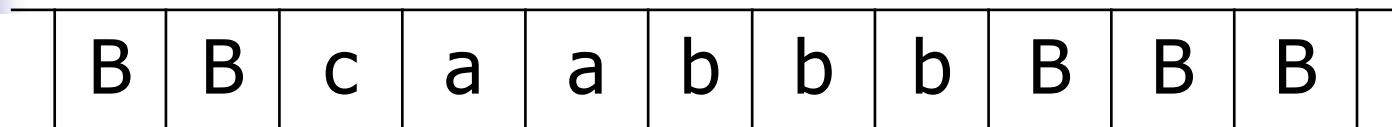
$\{a^i b^i \mid i \in \mathbb{N}\}$ を受理するTM (1)



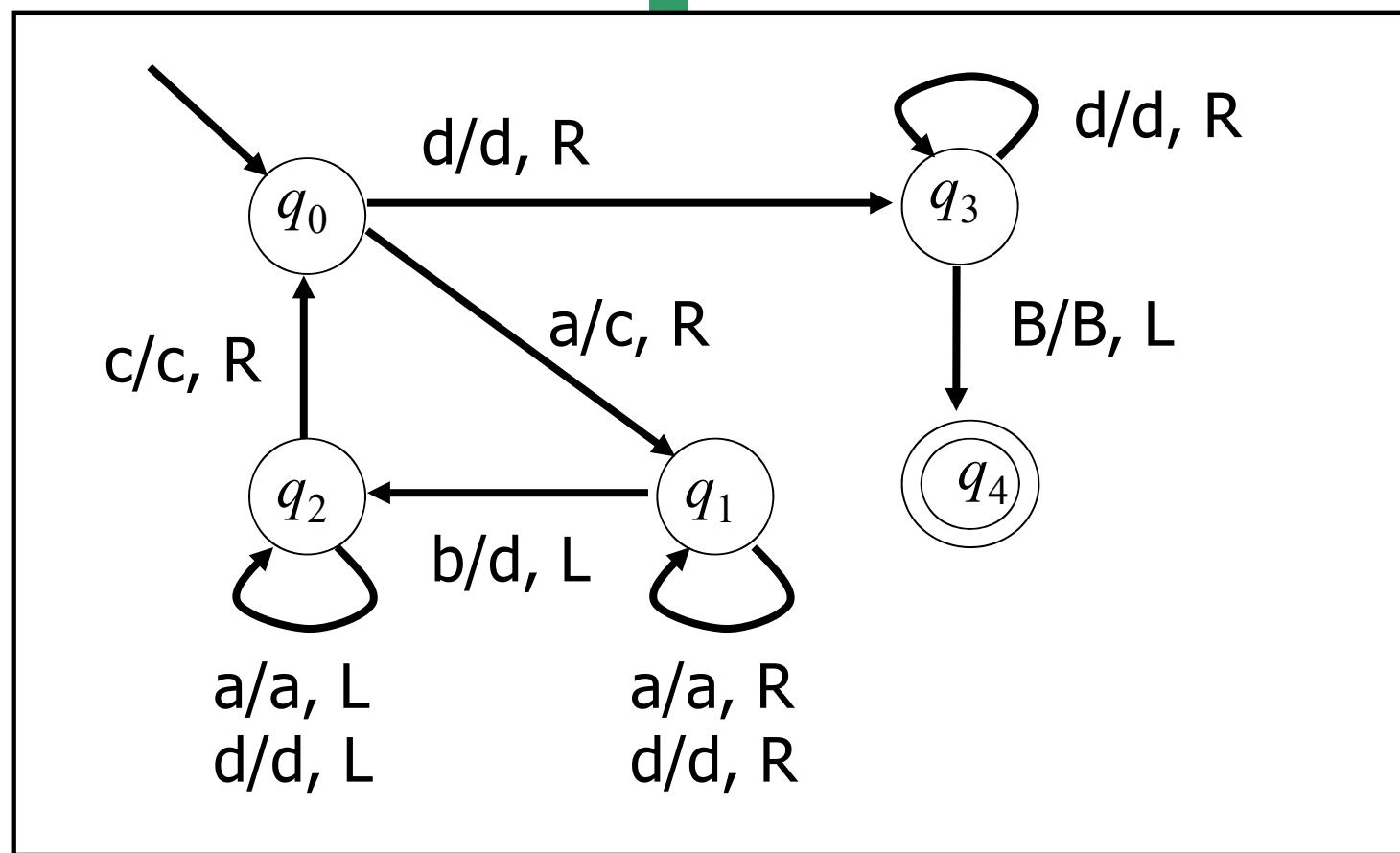
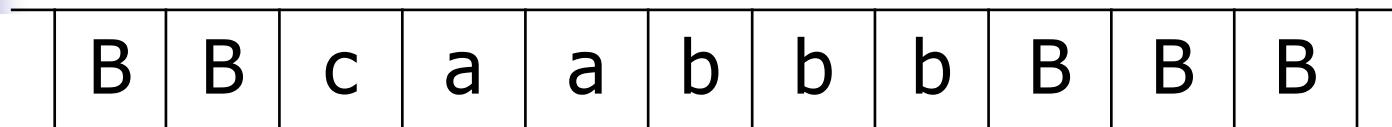
$\{a^i b^i \mid i \in \mathbb{N}\}$ を受理するTM (2)



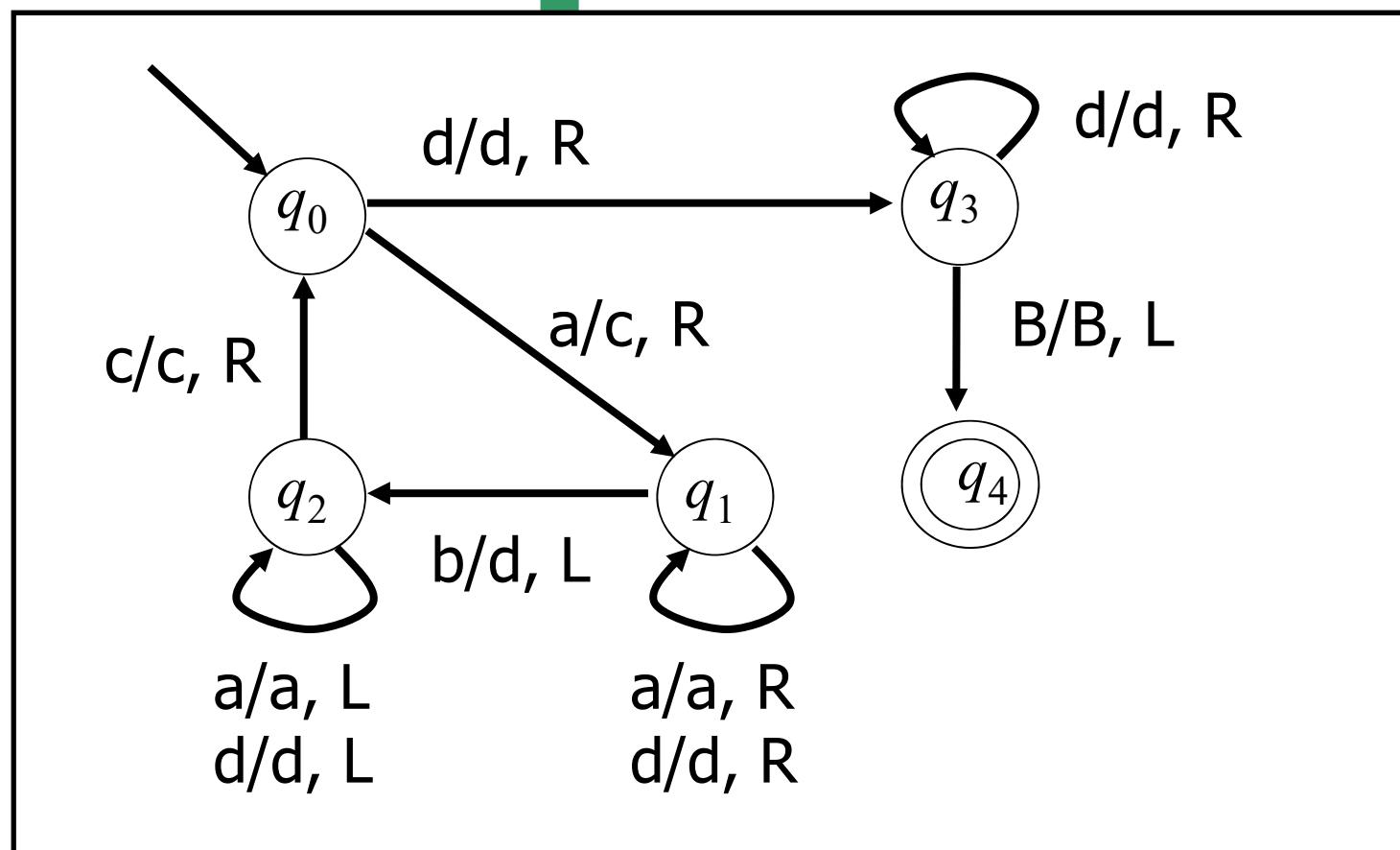
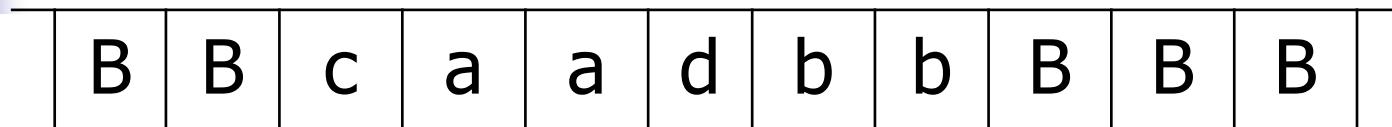
$\{a^i b^i \mid i \in \mathbb{N}\}$ を受理するTM (3)



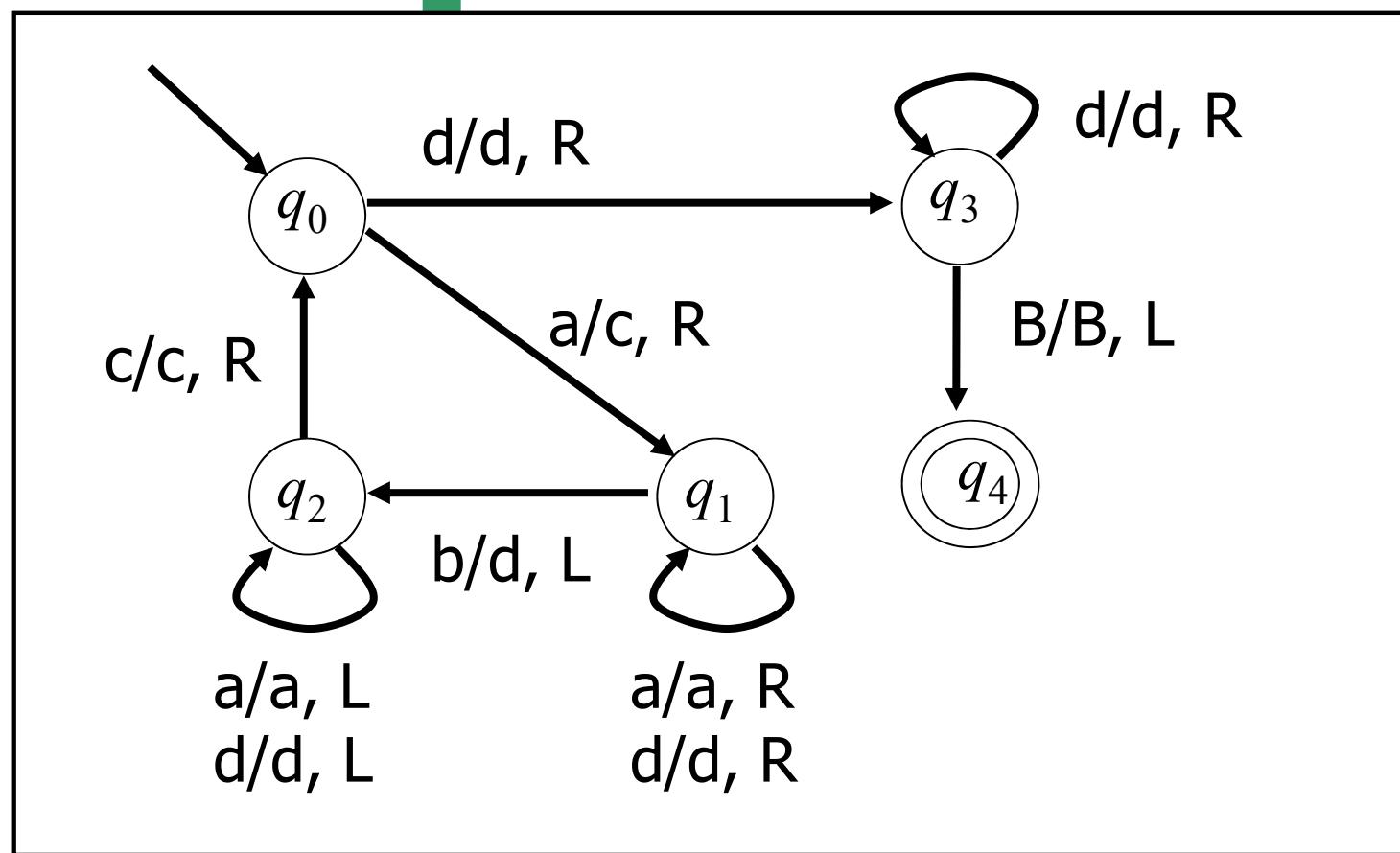
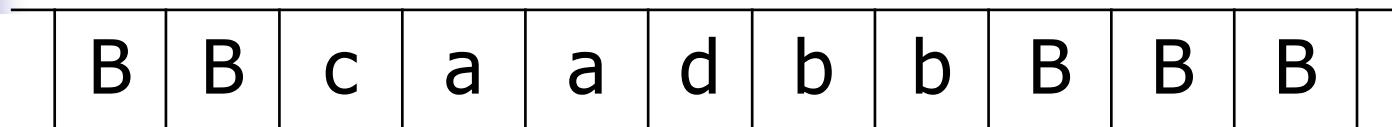
$\{a^i b^i \mid i \in \mathbb{N}\}$ を受理するTM (4)



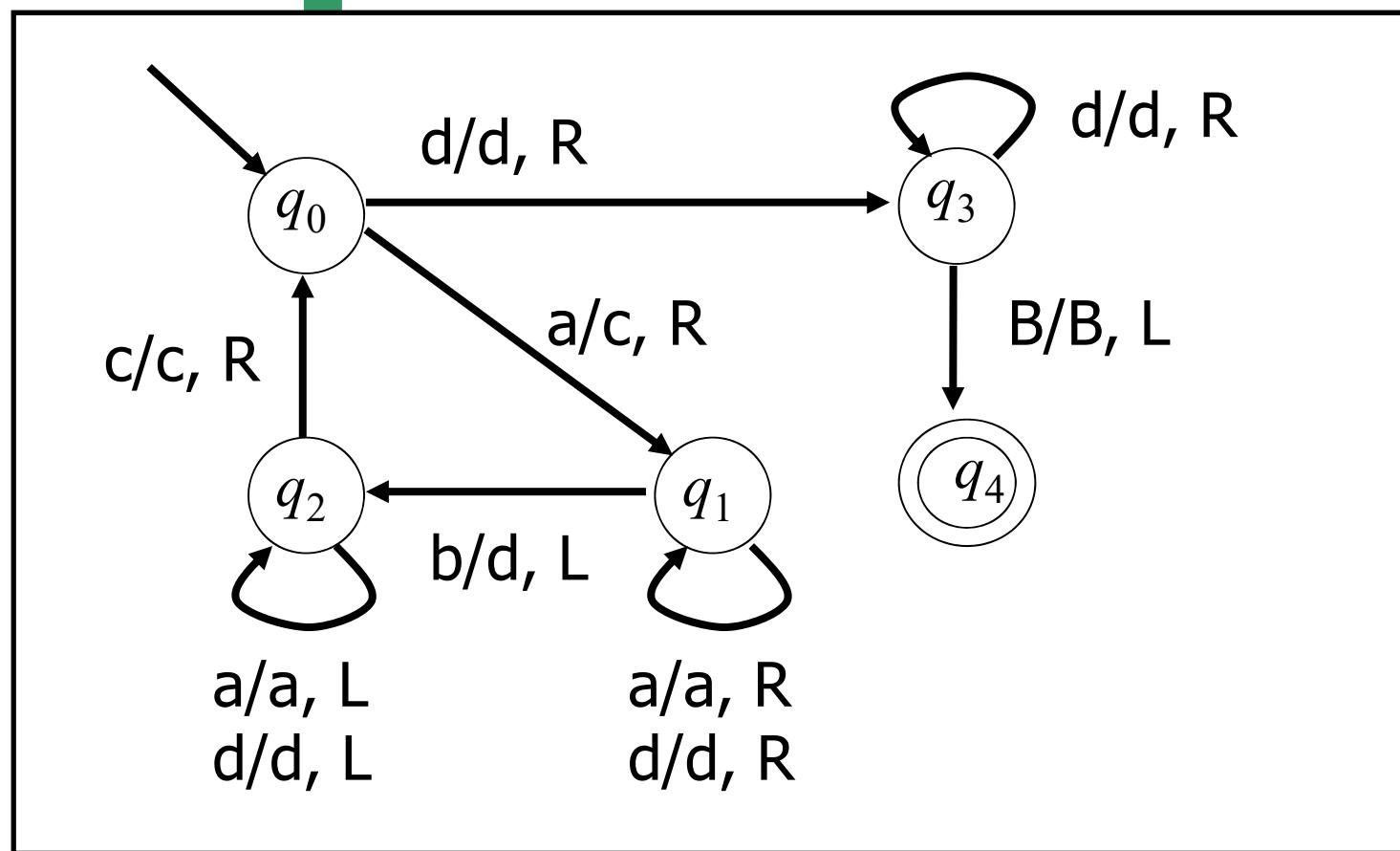
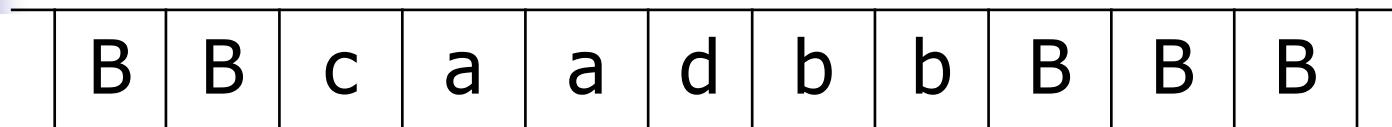
$\{a^i b^i \mid i \in \mathbb{N}\}$ を受理するTM (5)



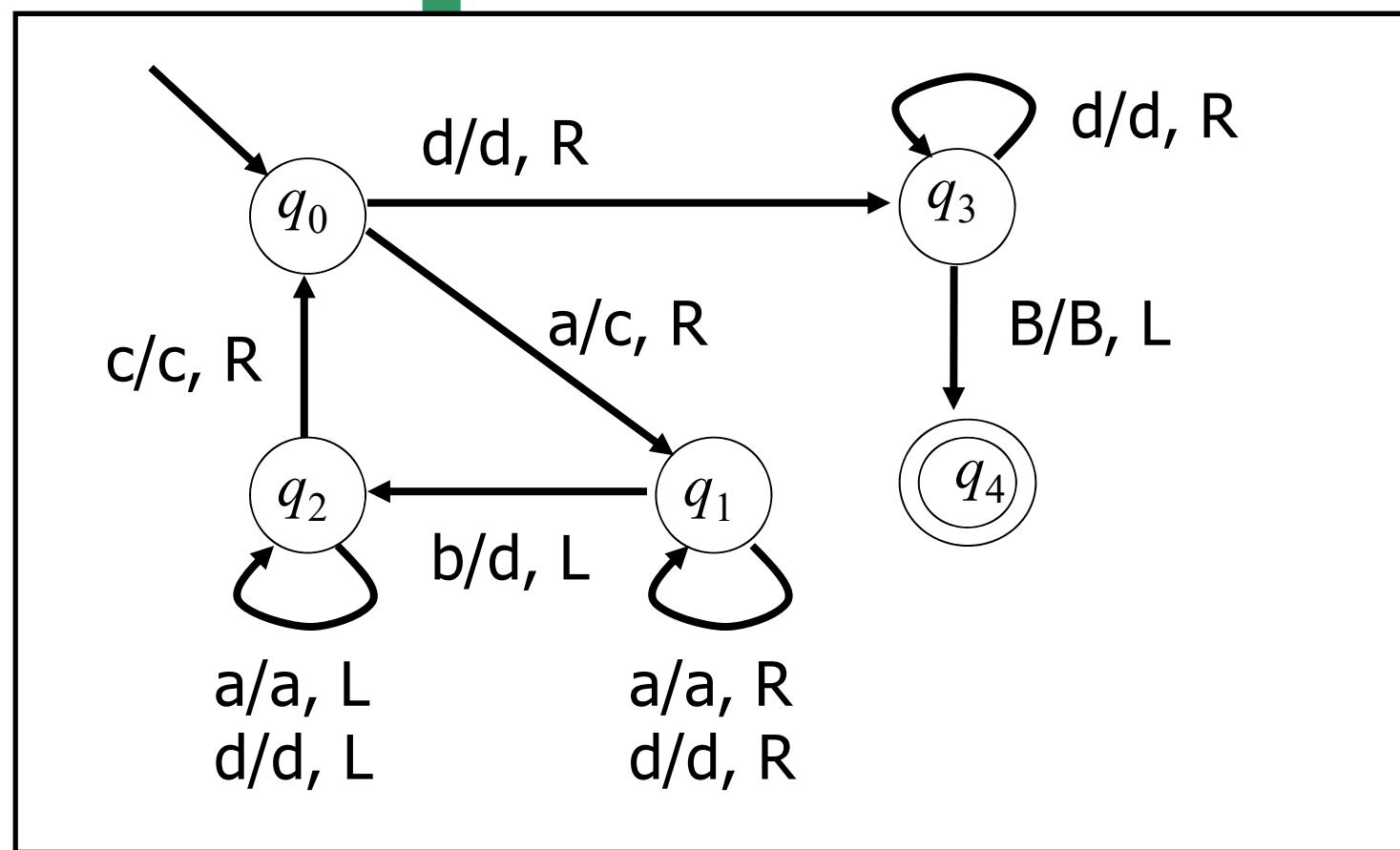
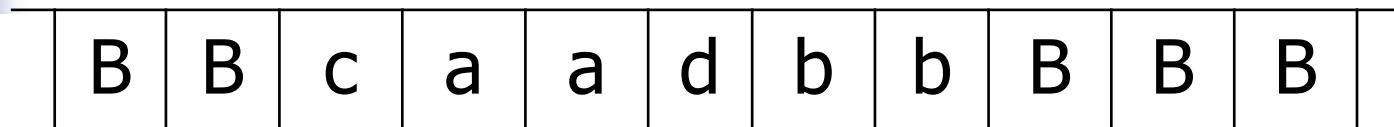
$\{a^i b^i \mid i \in \mathbb{N}\}$ を受理するTM (6)



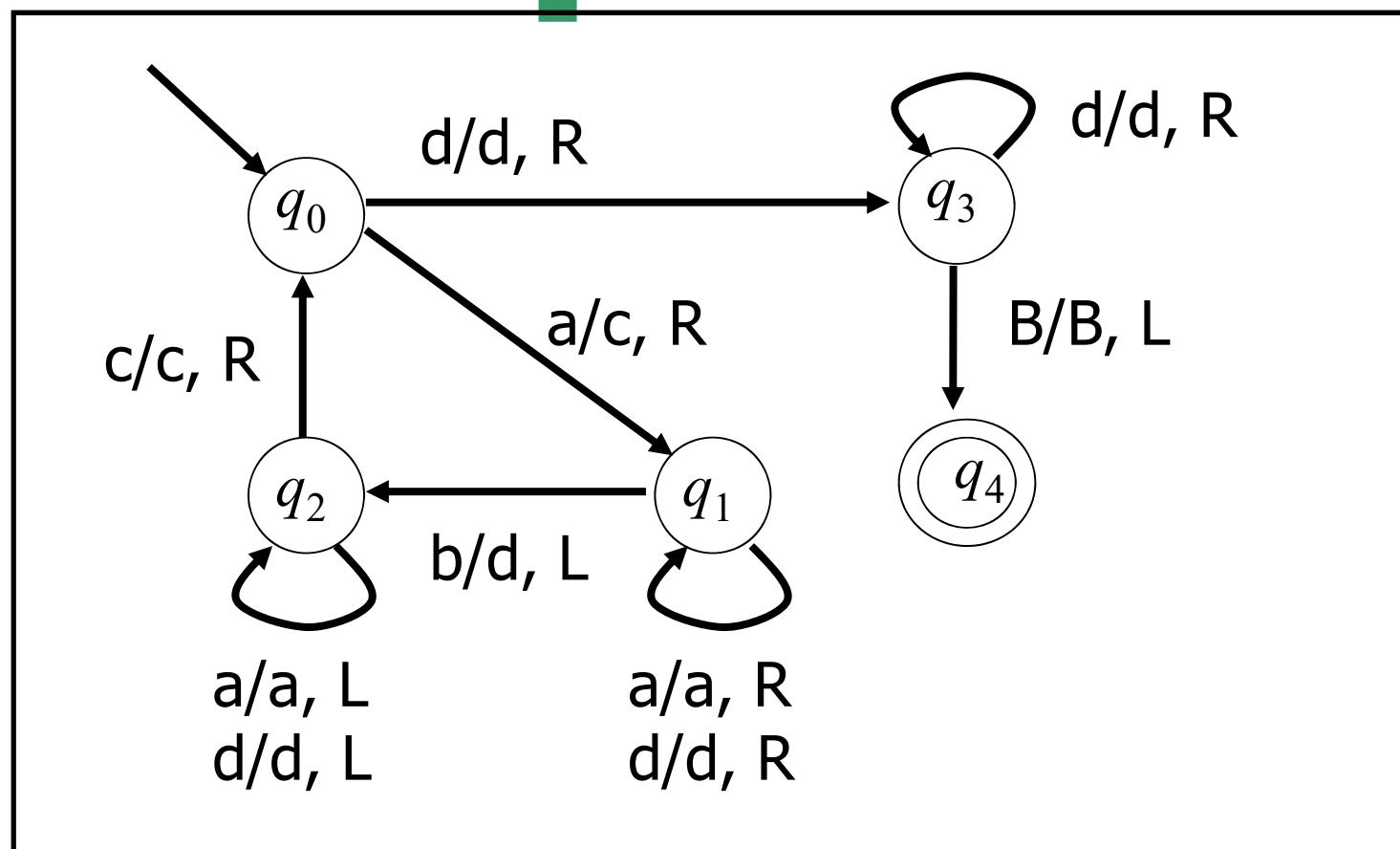
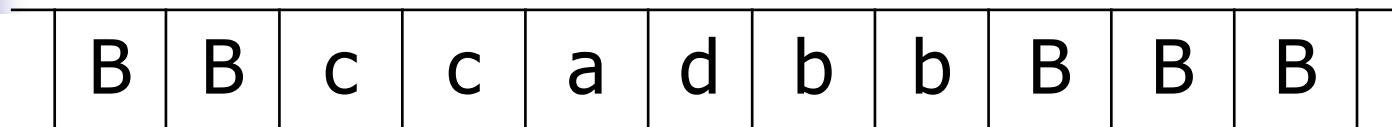
$\{a^i b^i \mid i \in \mathbb{N}\}$ を受理するTM (7)



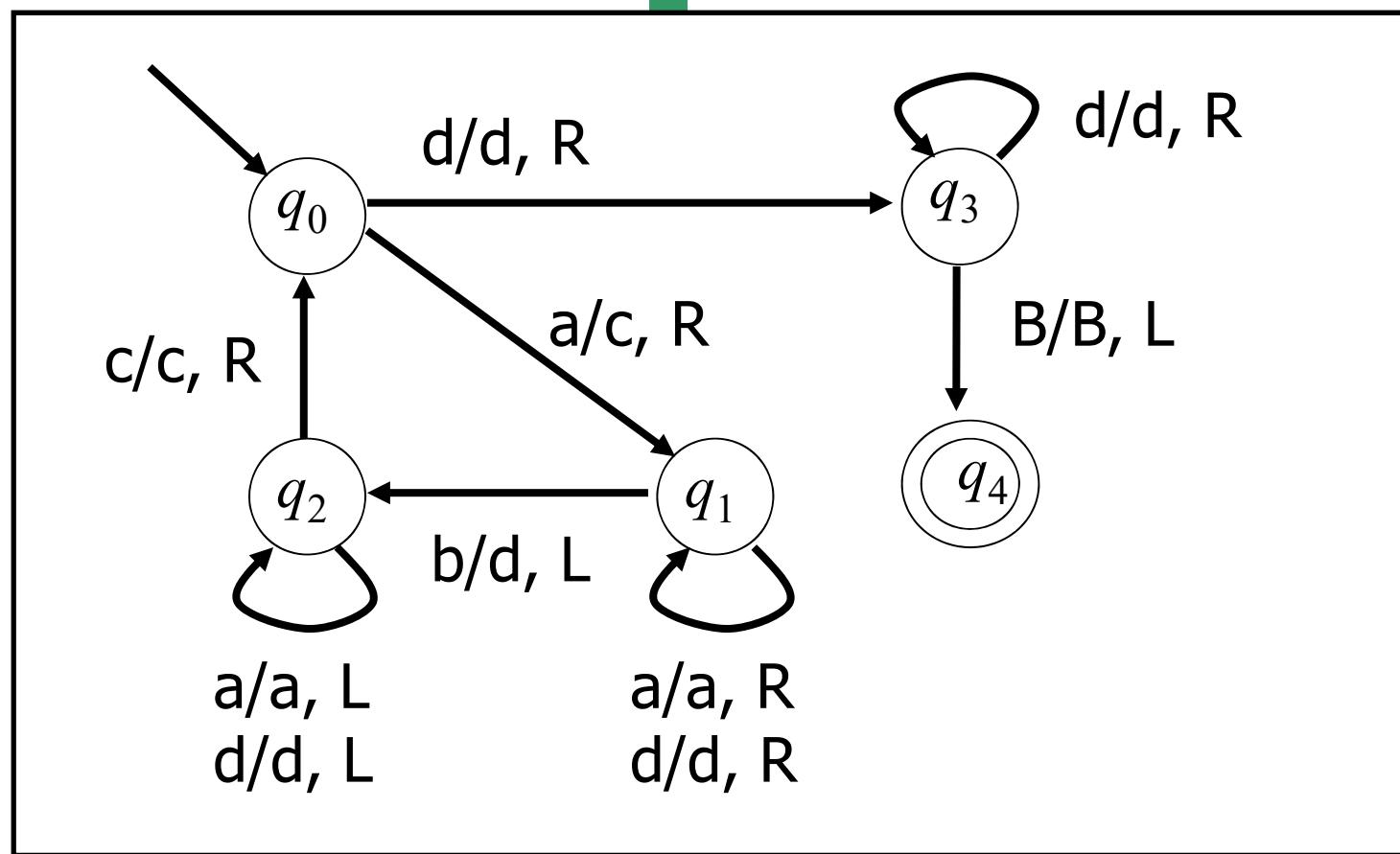
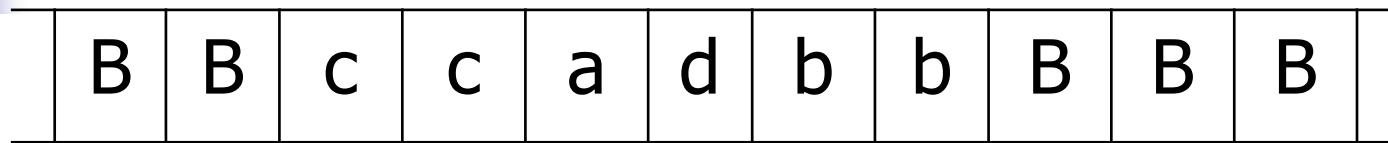
$\{a^i b^i \mid i \in \mathbb{N}\}$ を受理するTM (8)



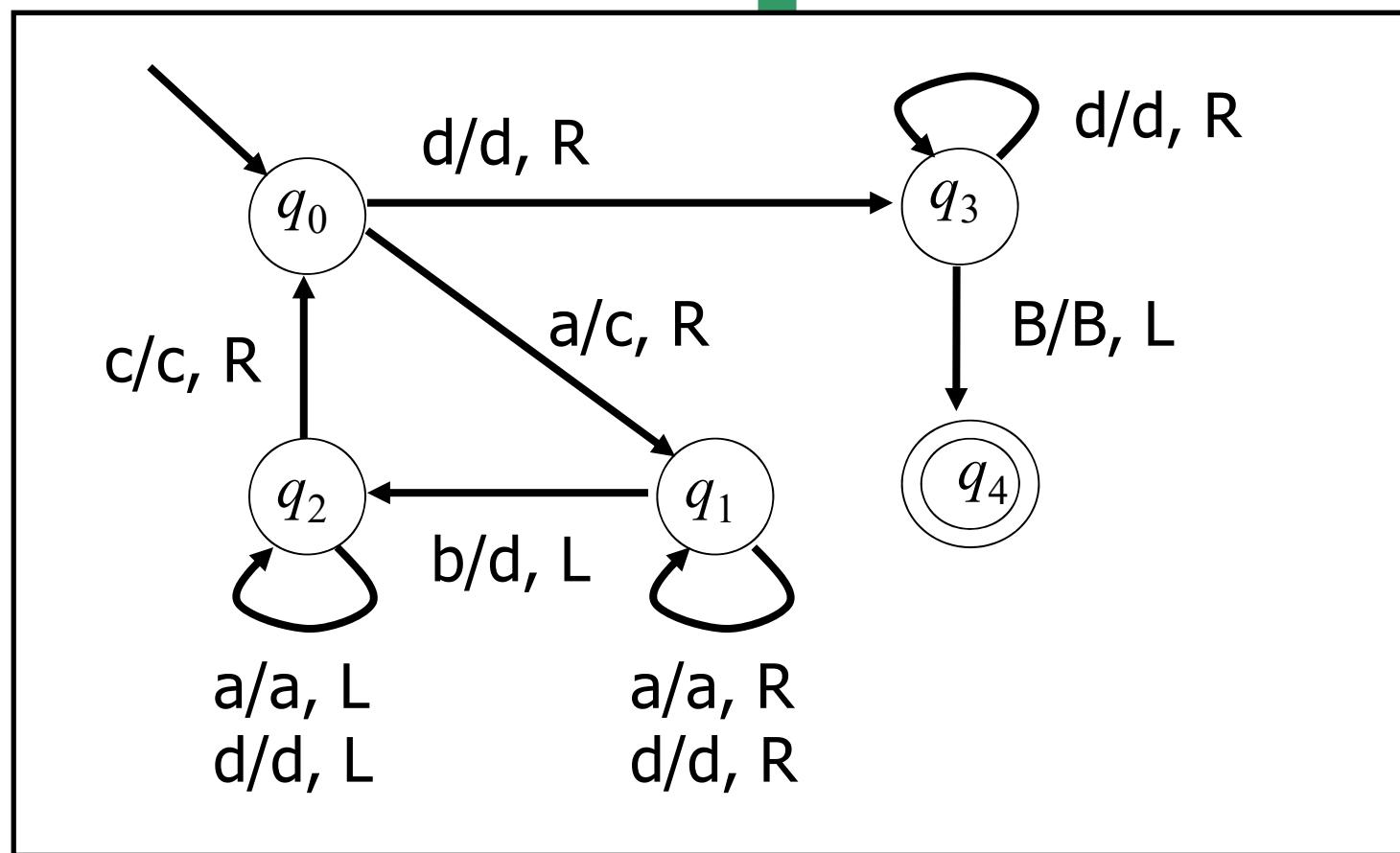
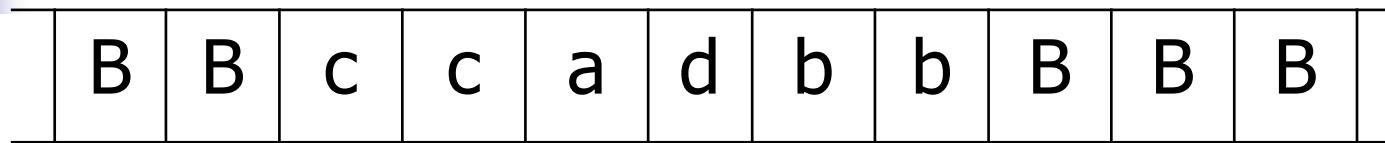
$\{a^i b^i \mid i \in \mathbb{N}\}$ を受理するTM (9)



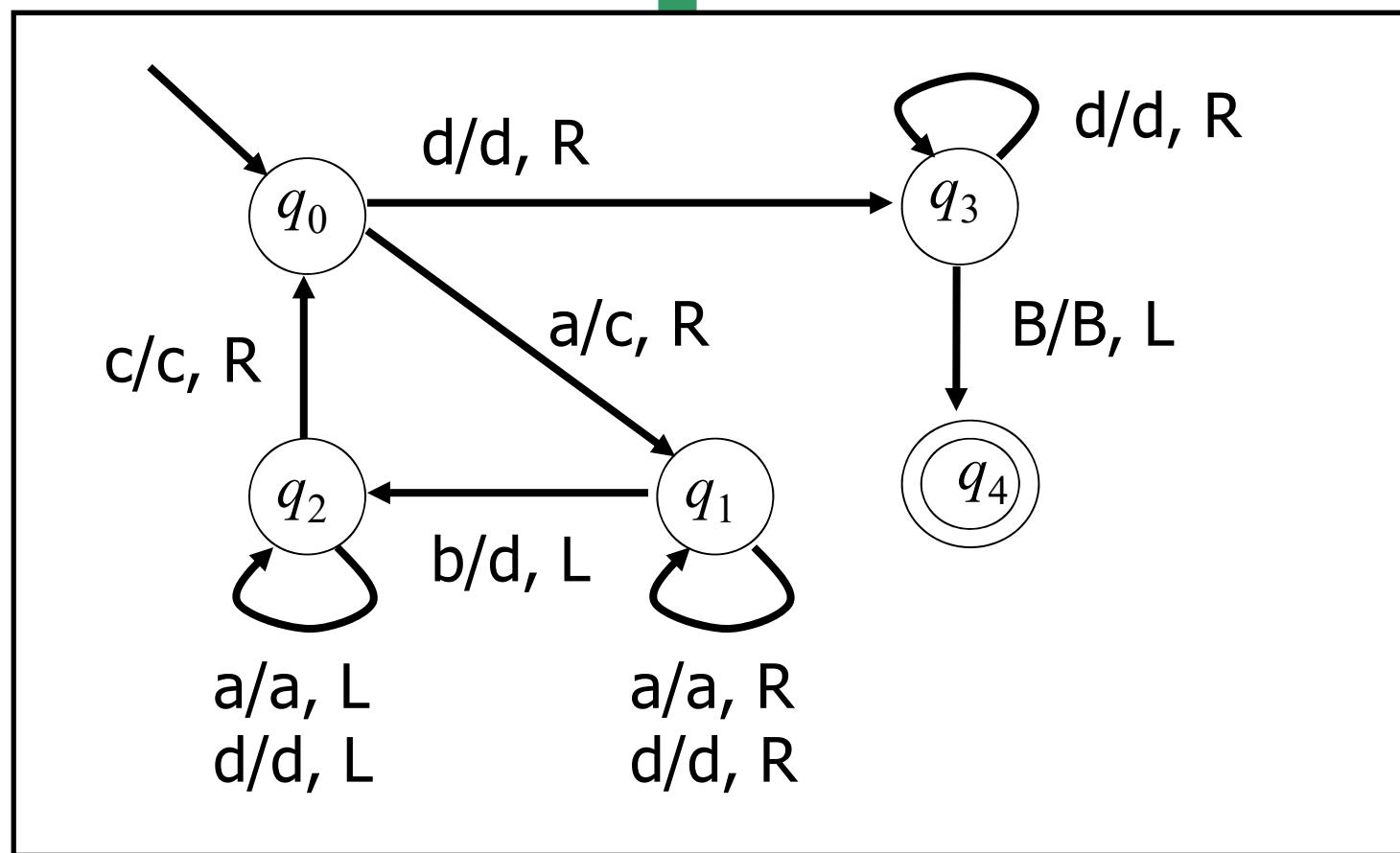
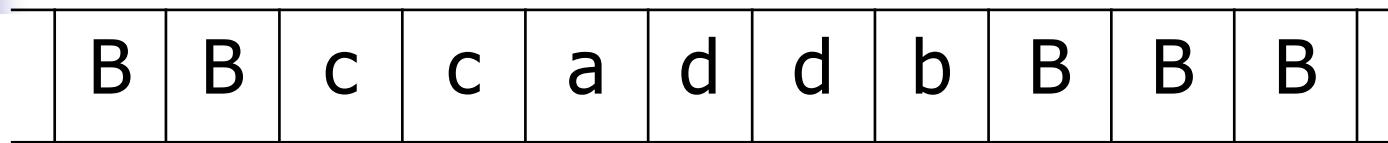
$\{a^i b^i \mid i \in \mathbb{N}\}$ を受理するTM (10)



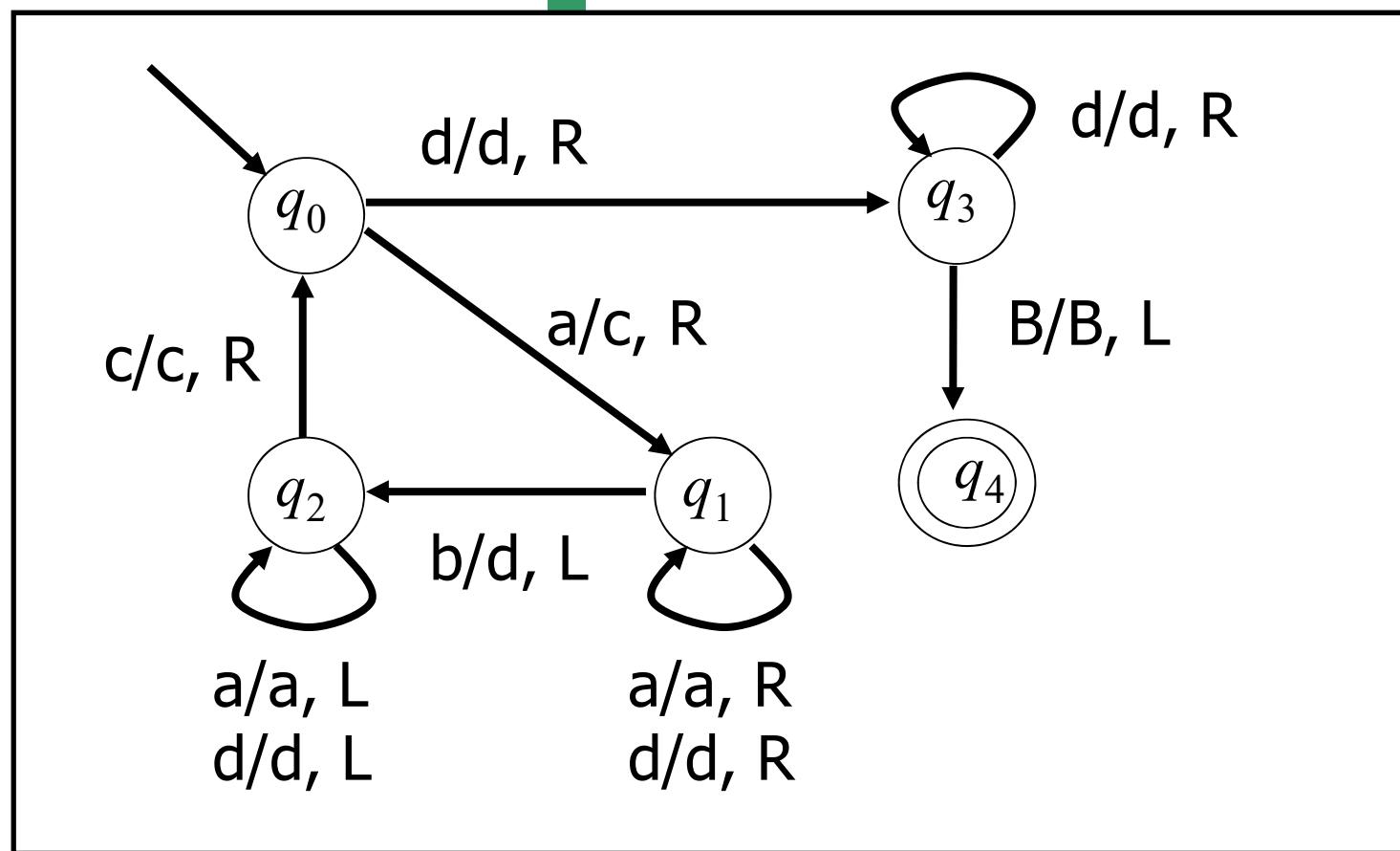
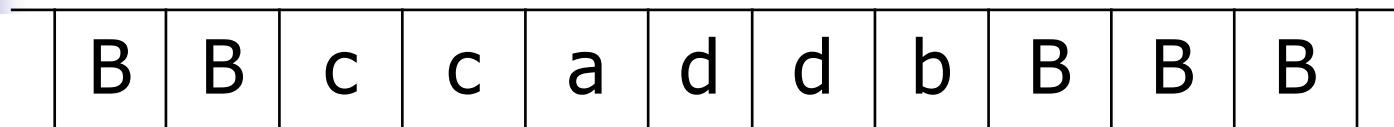
$\{a^i b^i \mid i \in \mathbb{N}\}$ を受理するTM (11)



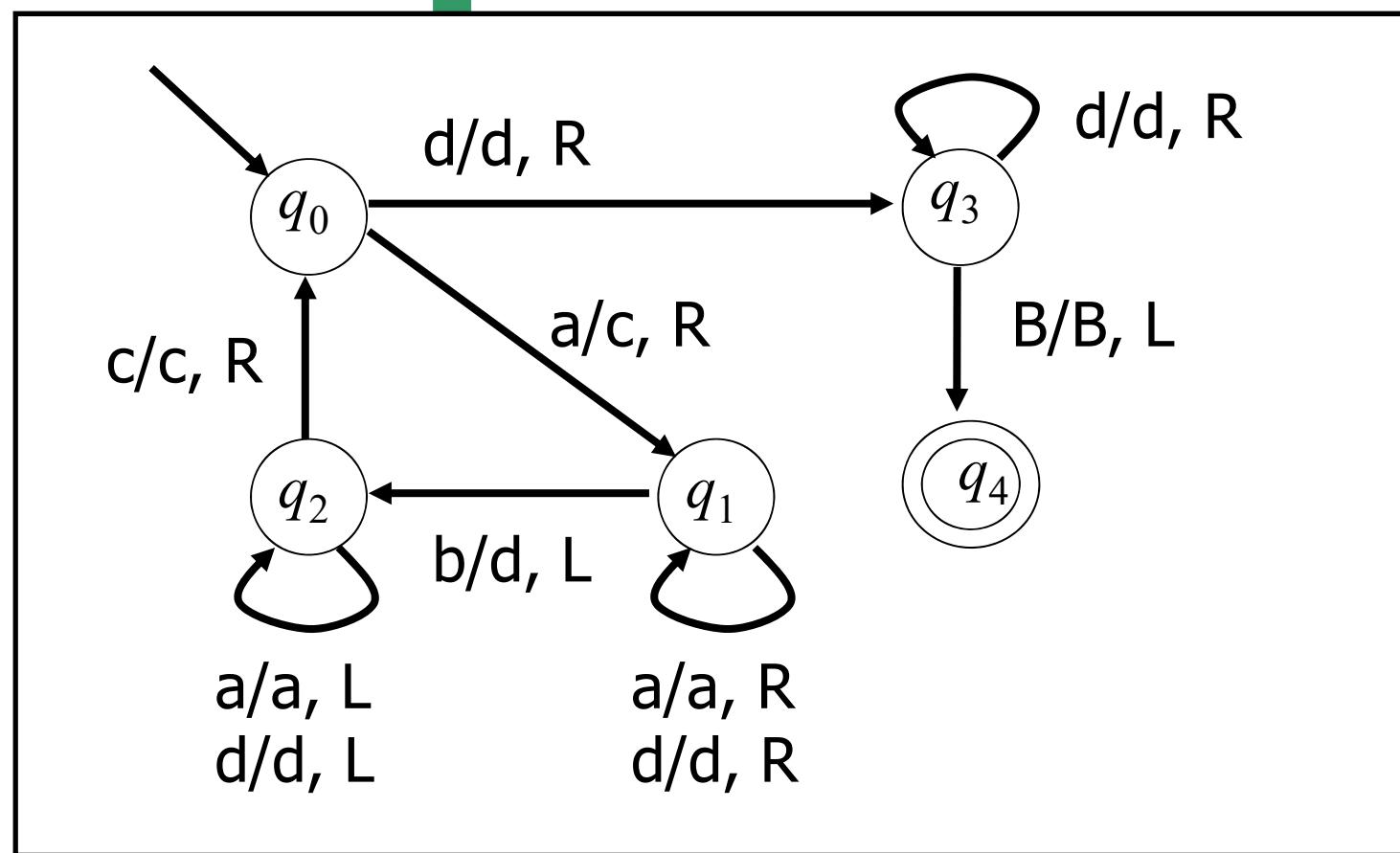
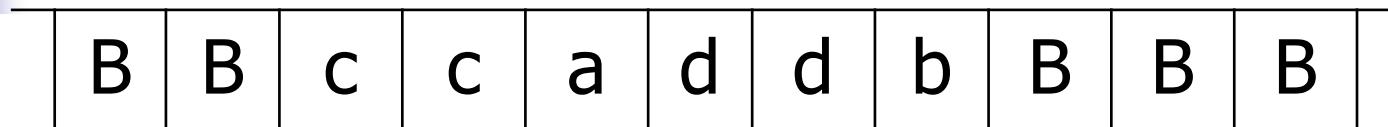
$\{a^i b^i \mid i \in \mathbb{N}\}$ を受理するTM (12)



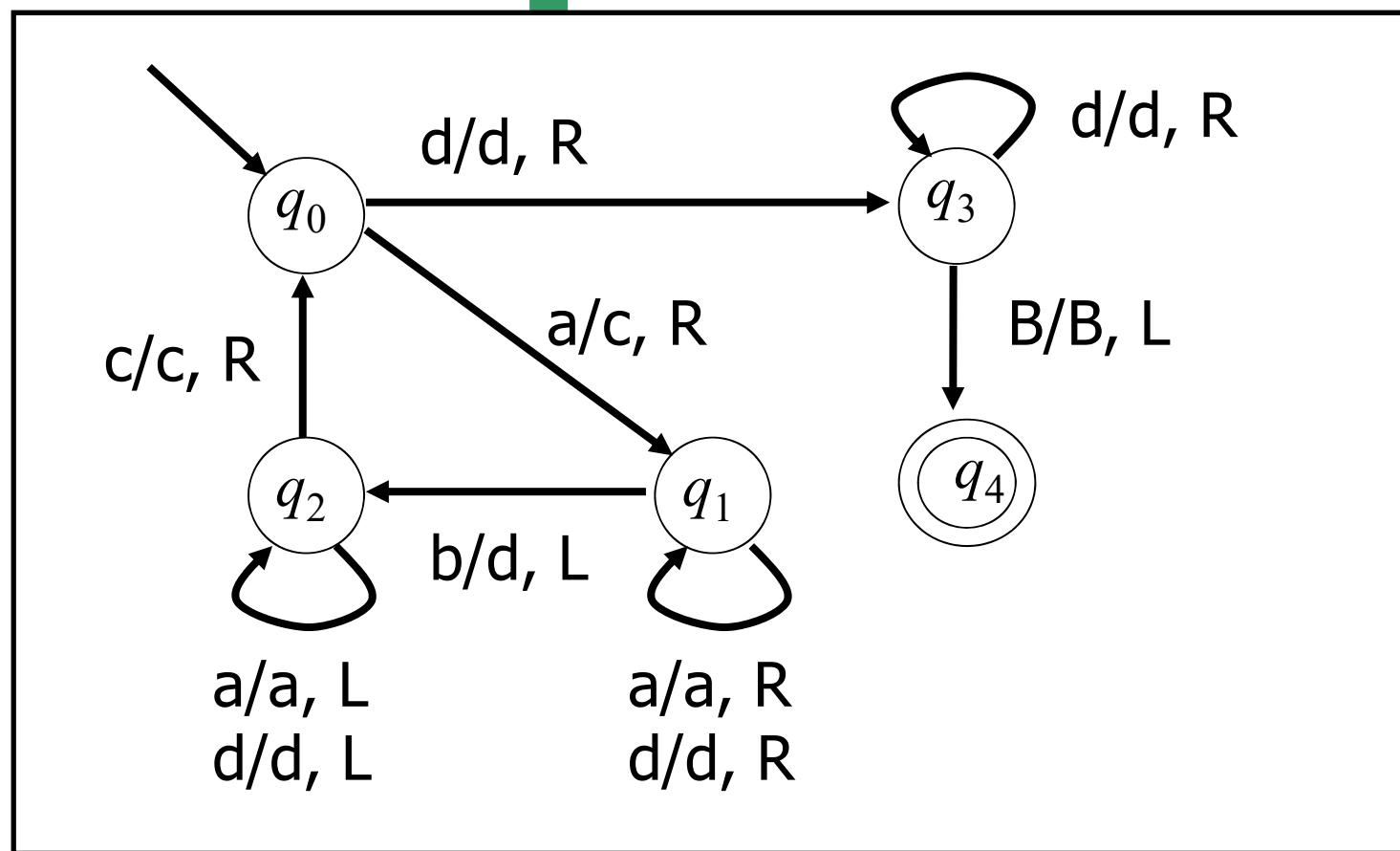
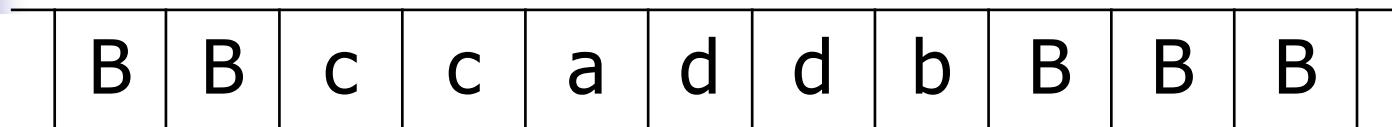
$\{a^i b^i \mid i \in \mathbb{N}\}$ を受理するTM (13)



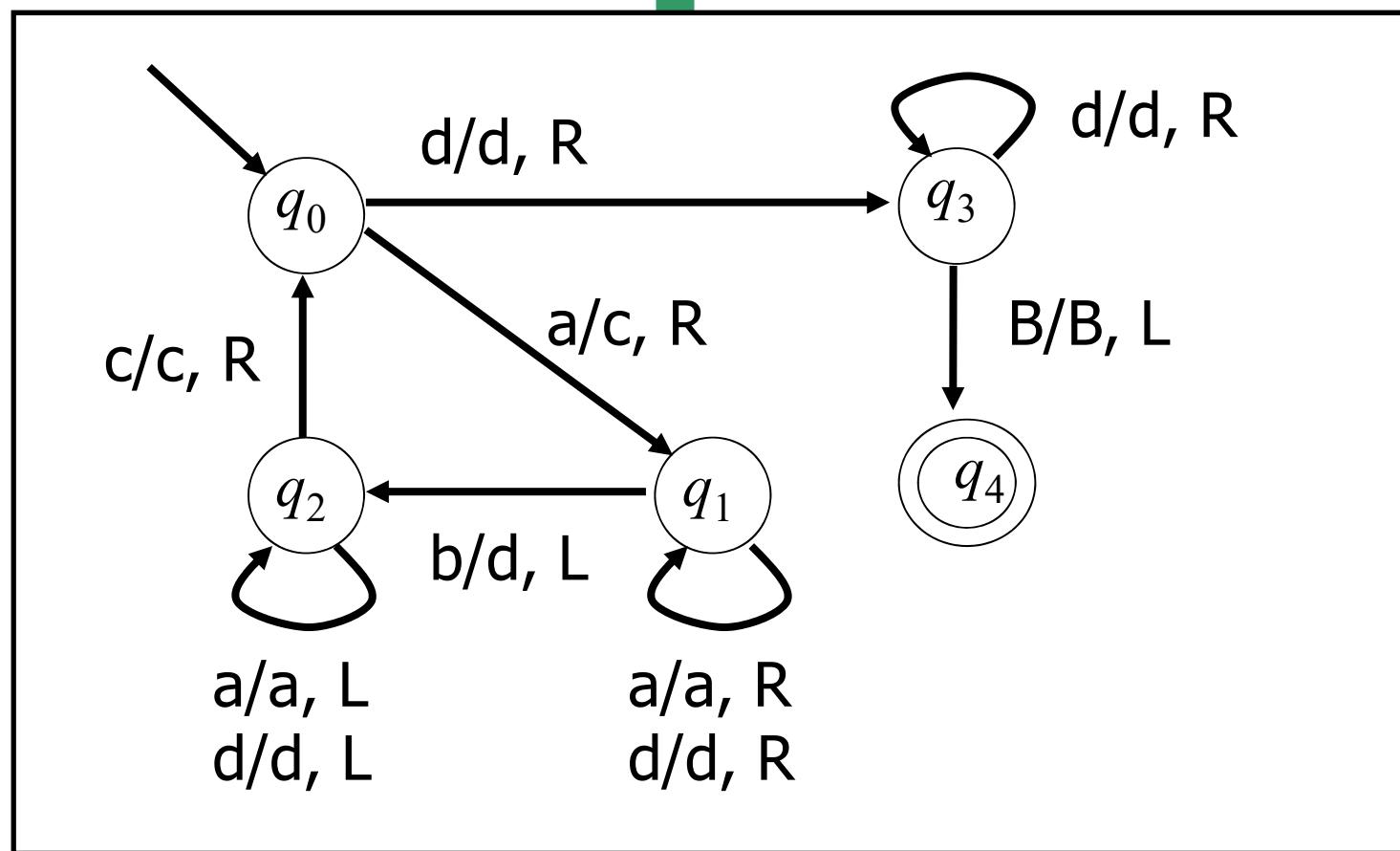
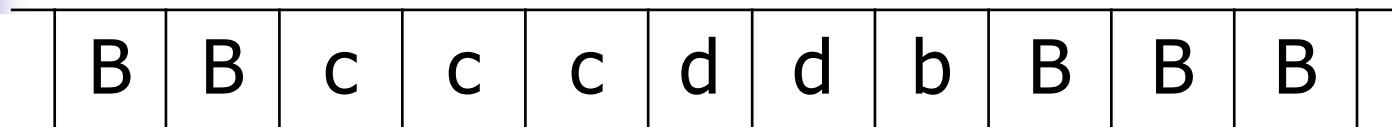
$\{a^i b^i \mid i \in \mathbb{N}\}$ を受理するTM (14)



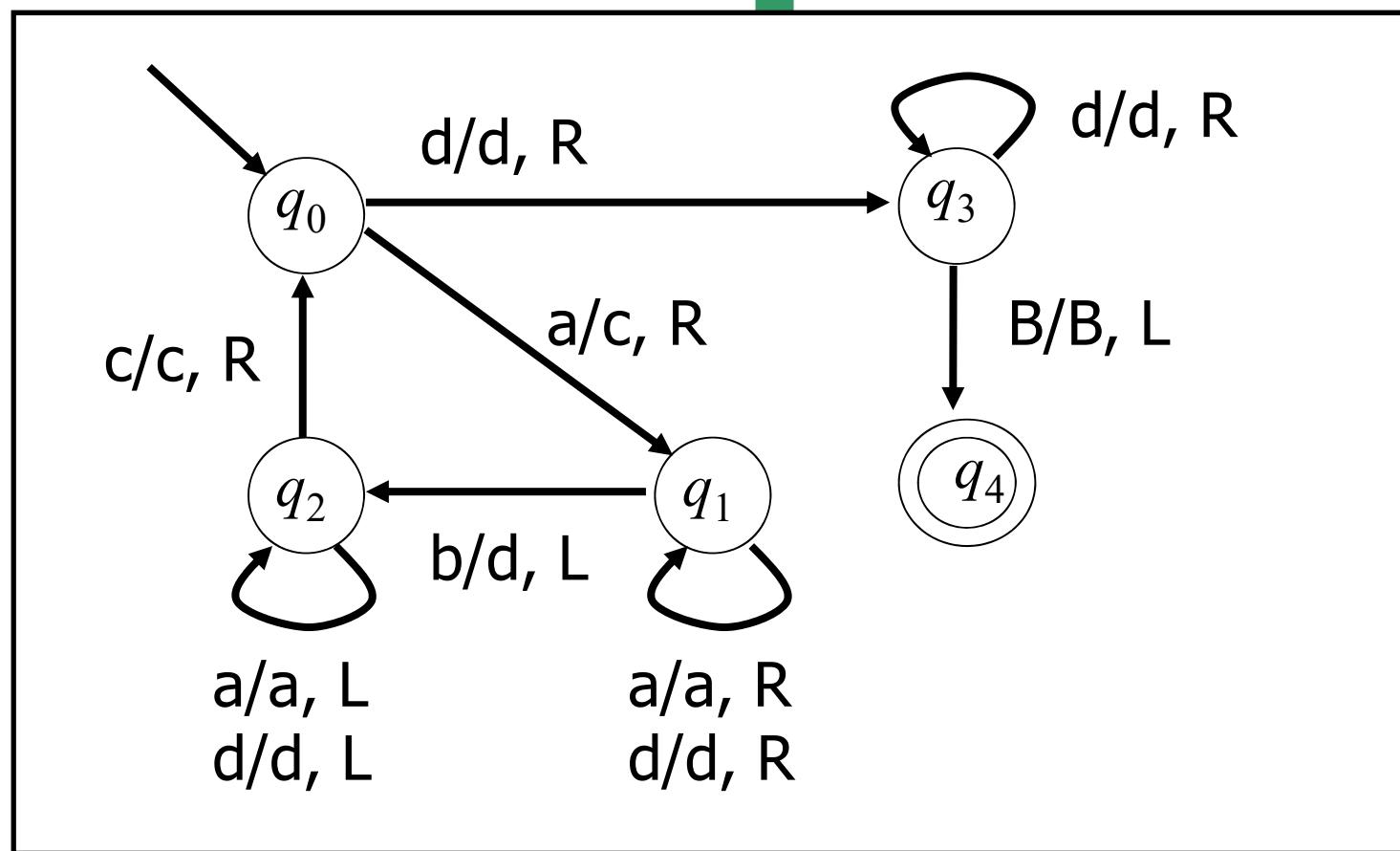
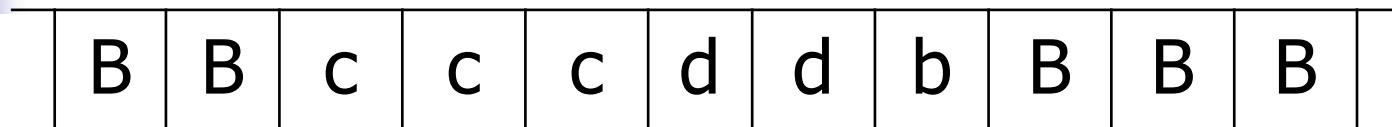
$\{a^i b^i \mid i \in \mathbb{N}\}$ を受理するTM (15)



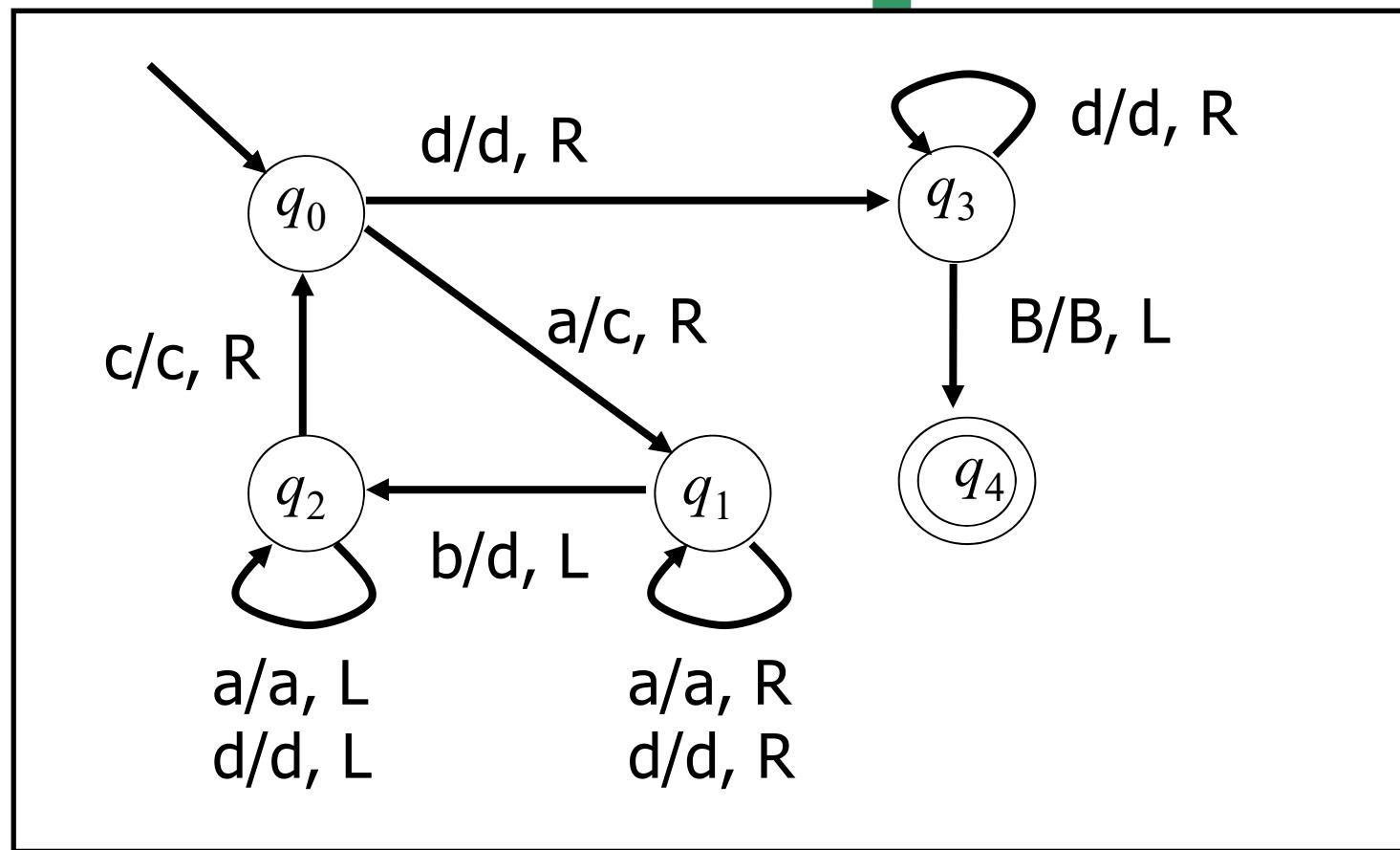
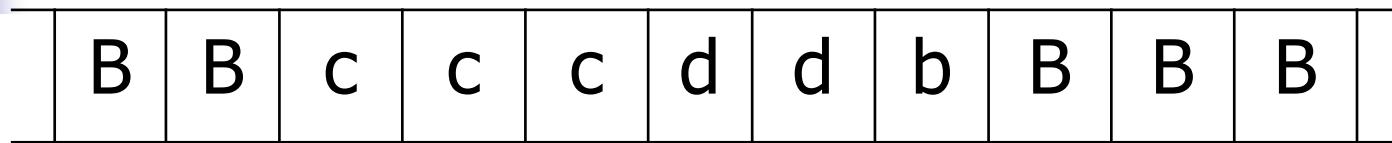
$\{a^i b^i \mid i \in \mathbb{N}\}$ を受理するTM (16)



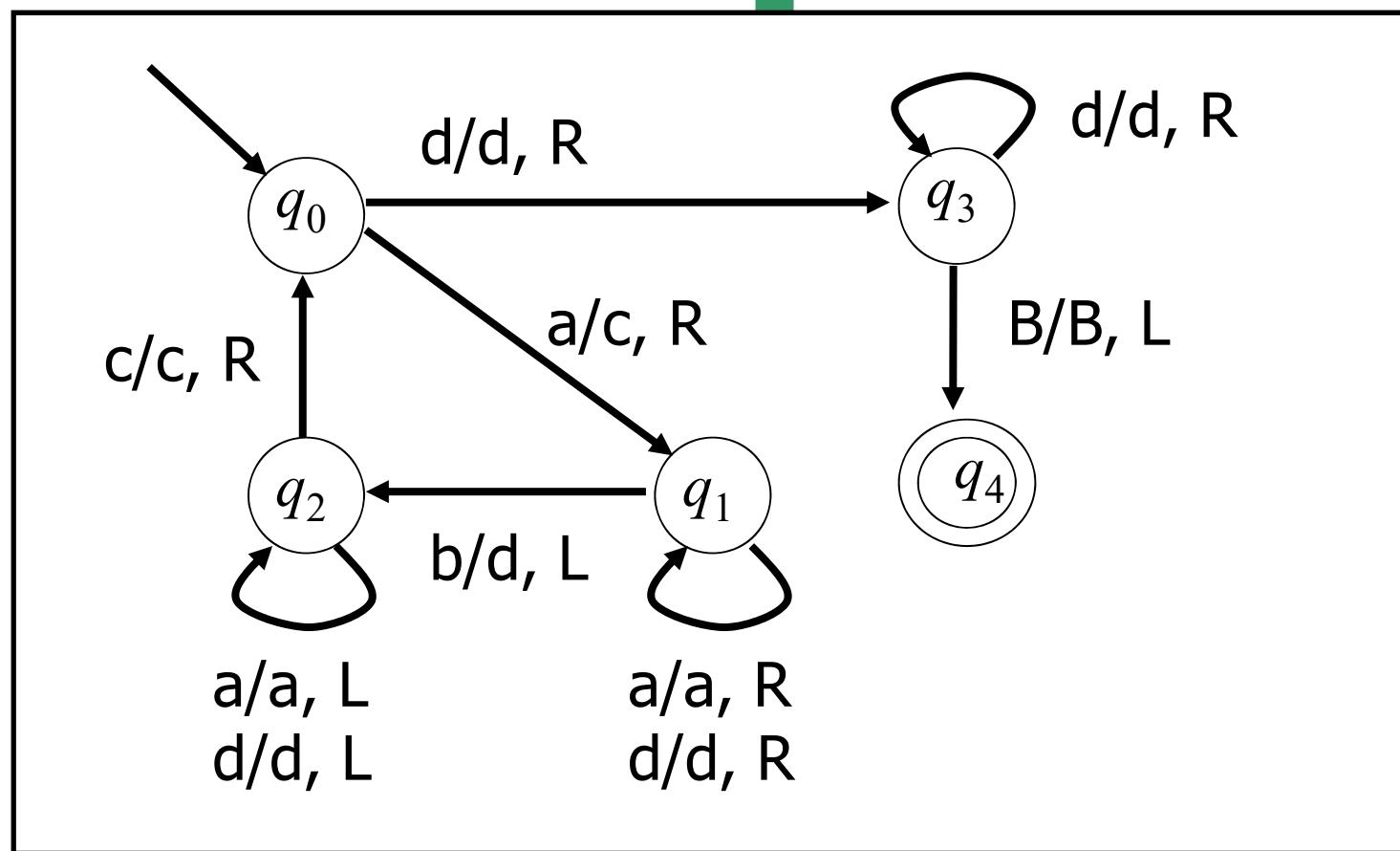
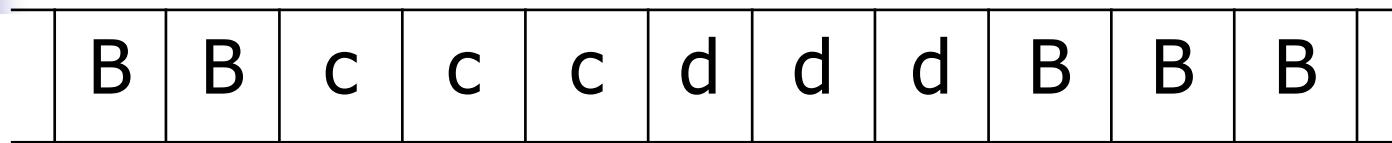
$\{a^i b^i \mid i \in \mathbb{N}\}$ を受理するTM (17)



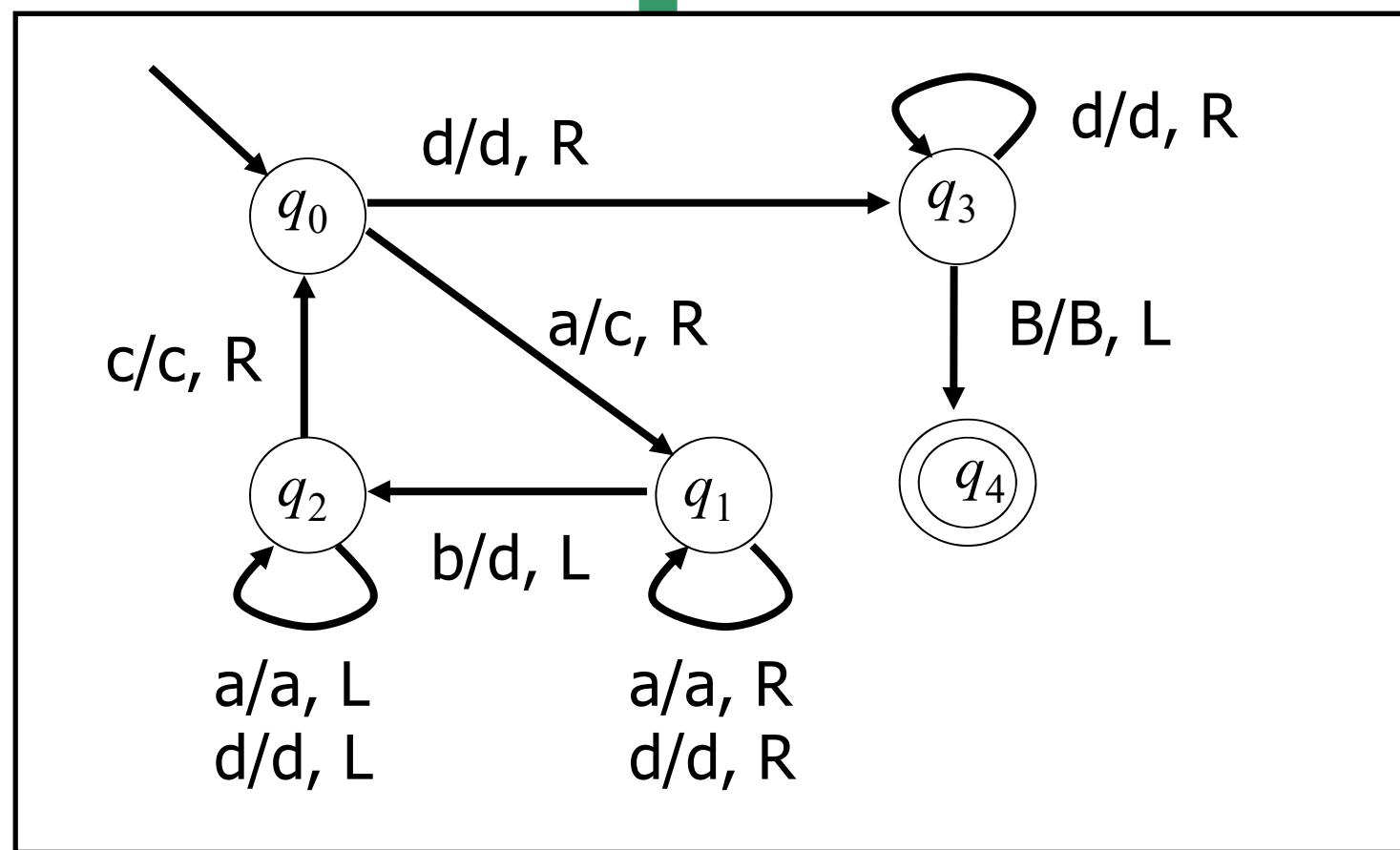
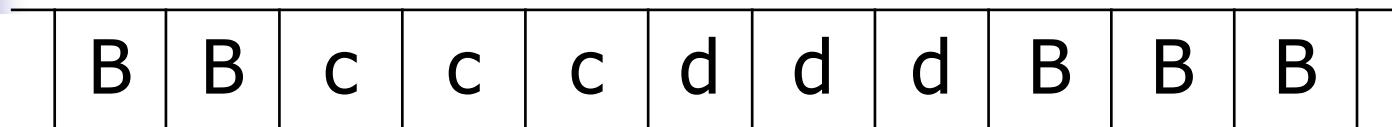
$\{a^i b^i \mid i \in \mathbb{N}\}$ を受理するTM (18)



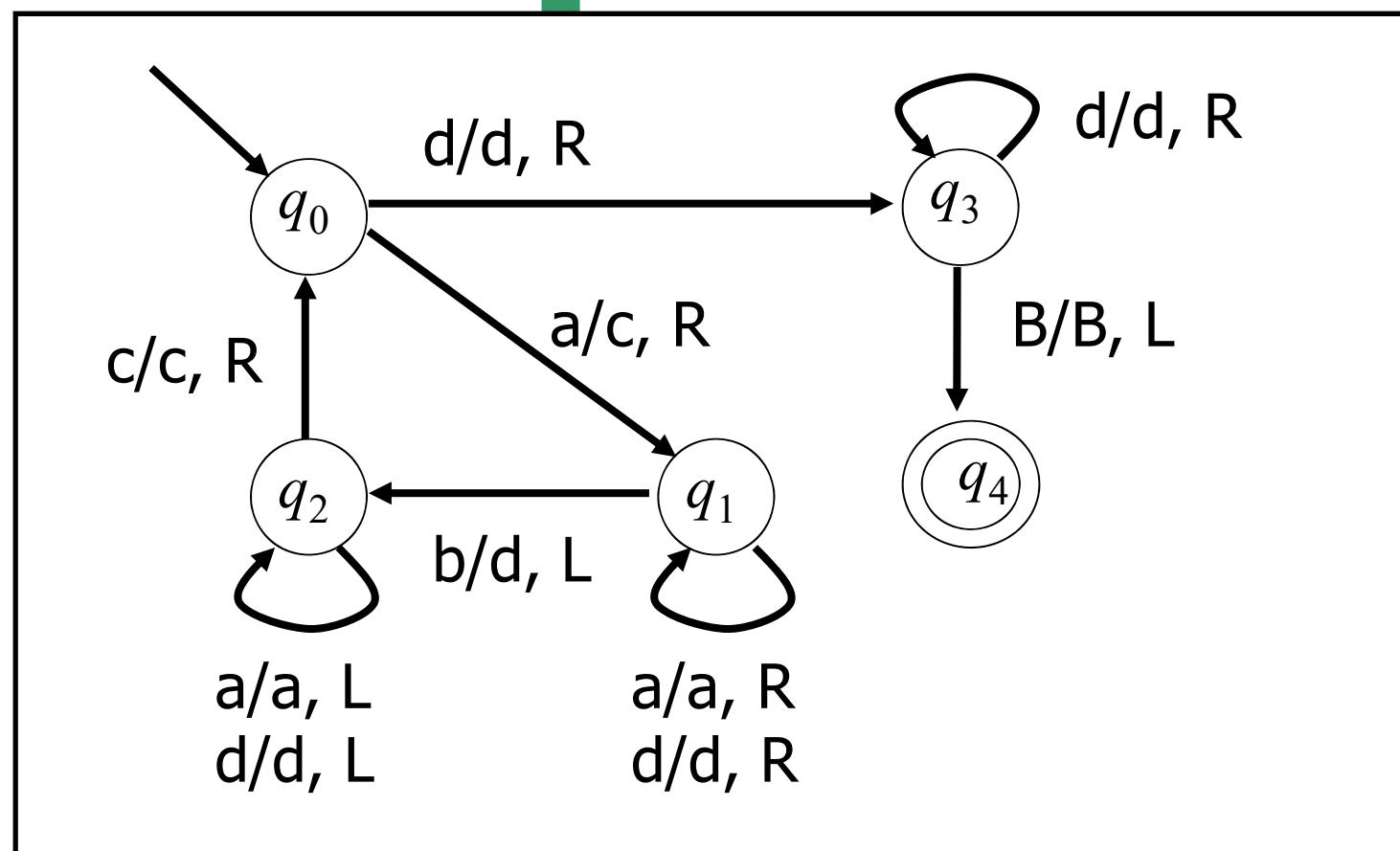
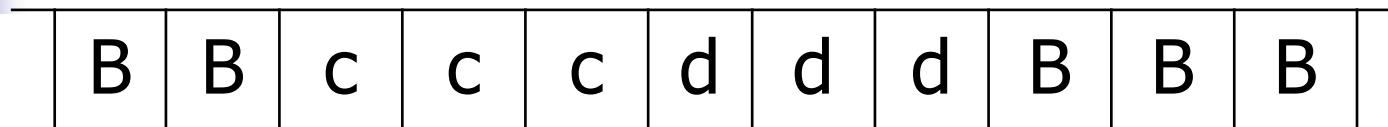
$\{a^i b^i \mid i \in \mathbb{N}\}$ を受理するTM (19)



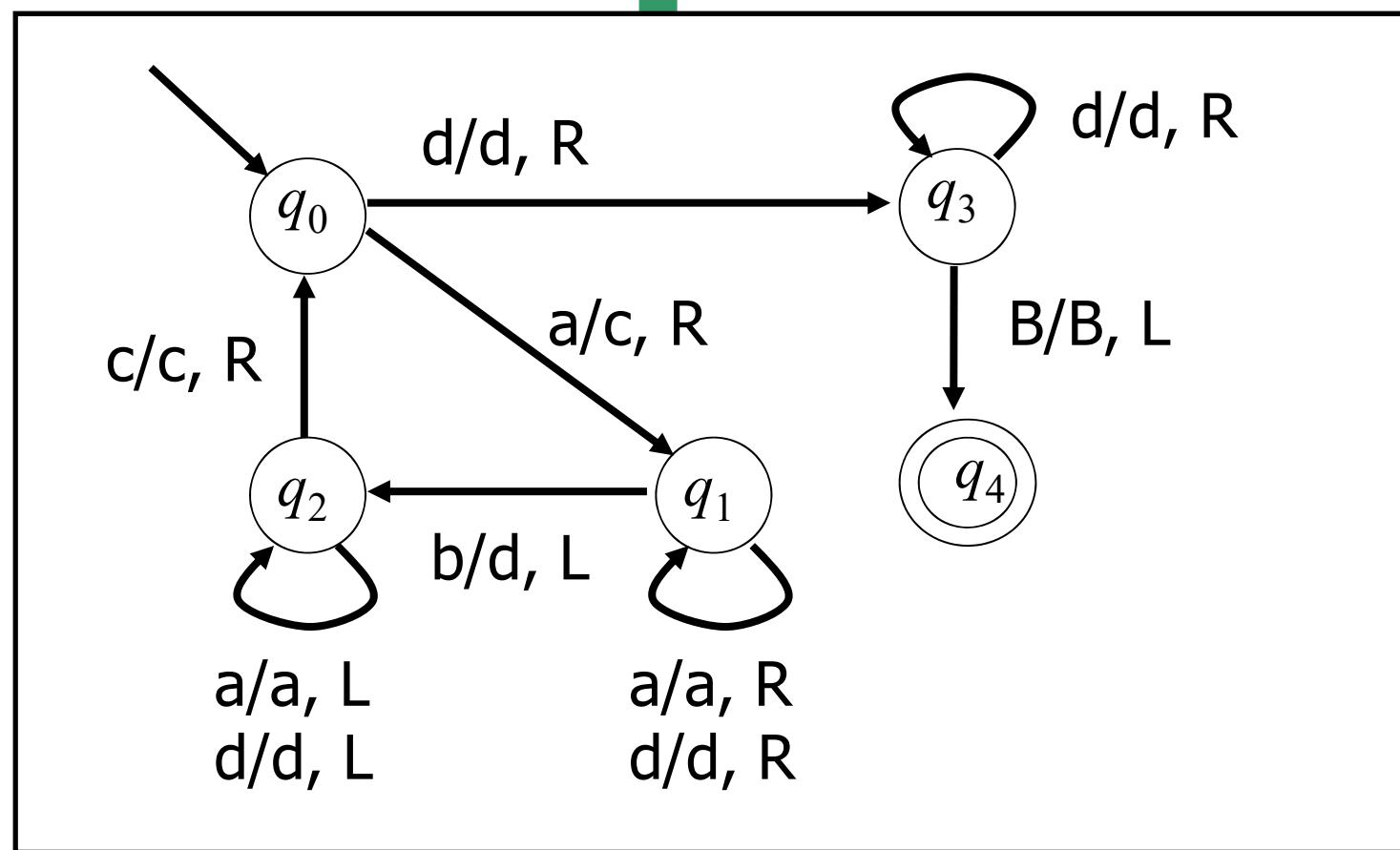
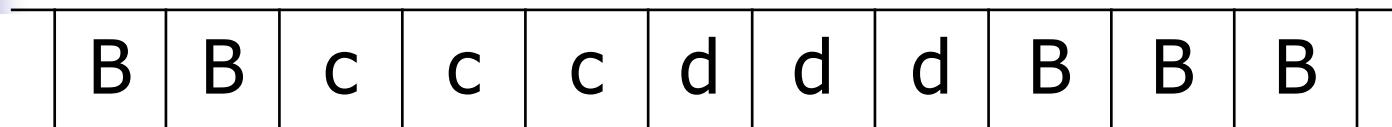
$\{a^i b^i \mid i \in \mathbb{N}\}$ を受理するTM (20)



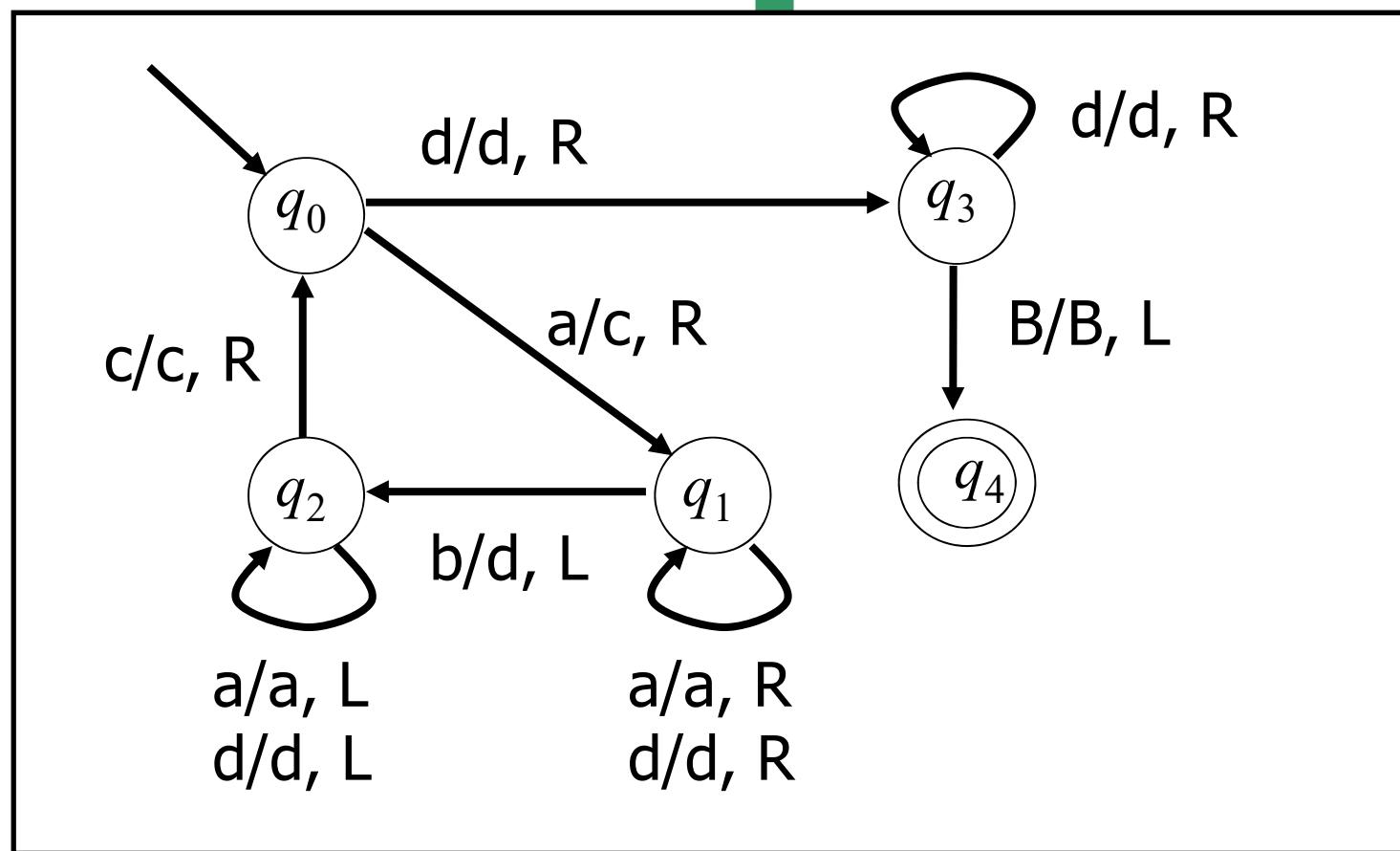
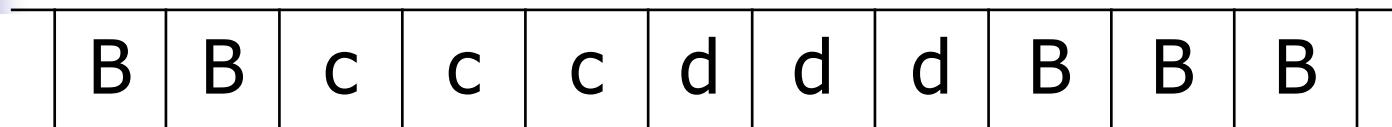
$\{a^i b^i \mid i \in \mathbb{N}\}$ を受理するTM (21)



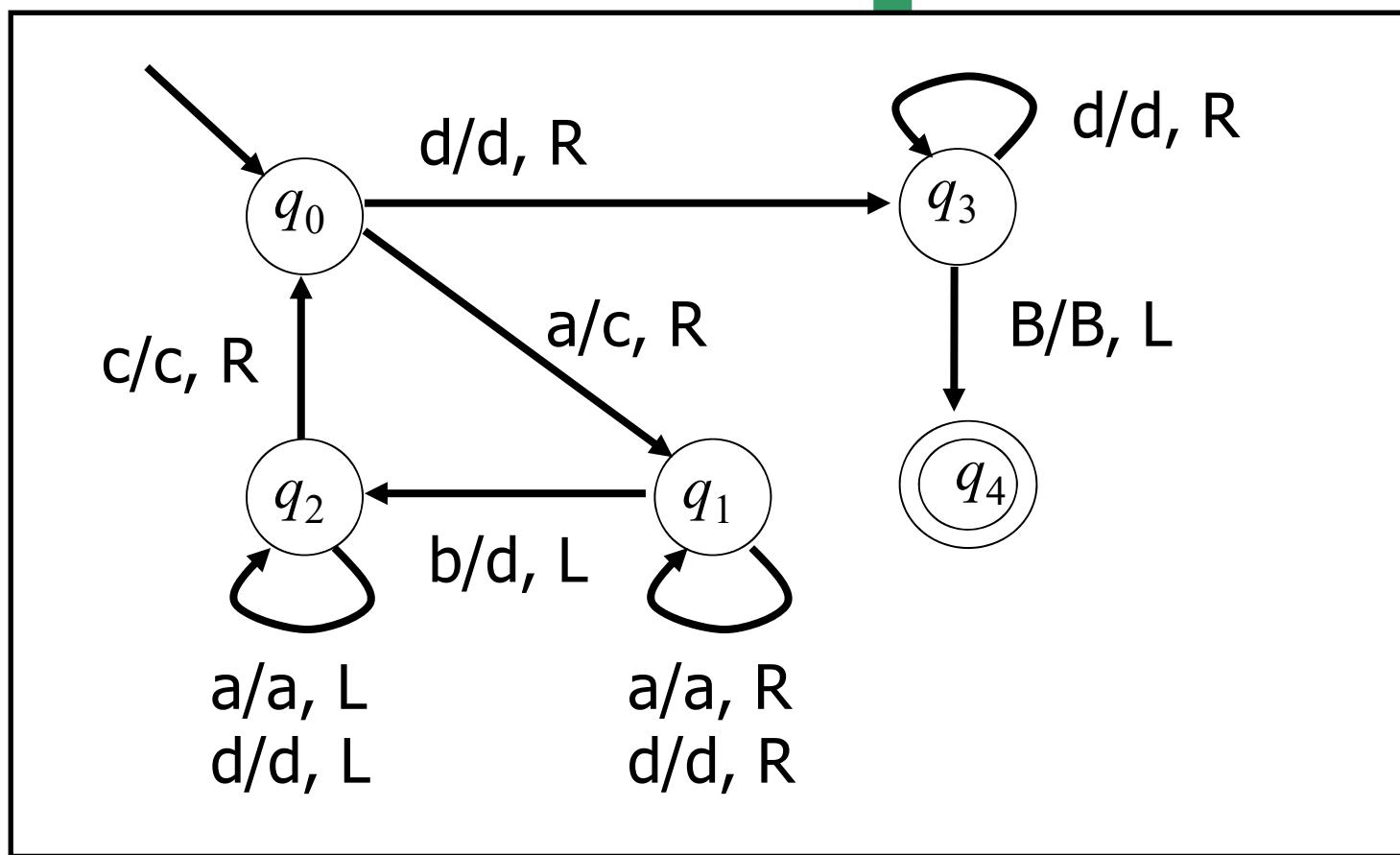
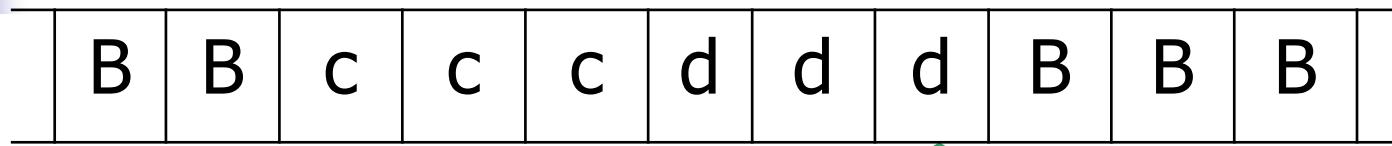
$\{a^i b^i \mid i \in \mathbb{N}\}$ を受理するTM (22)



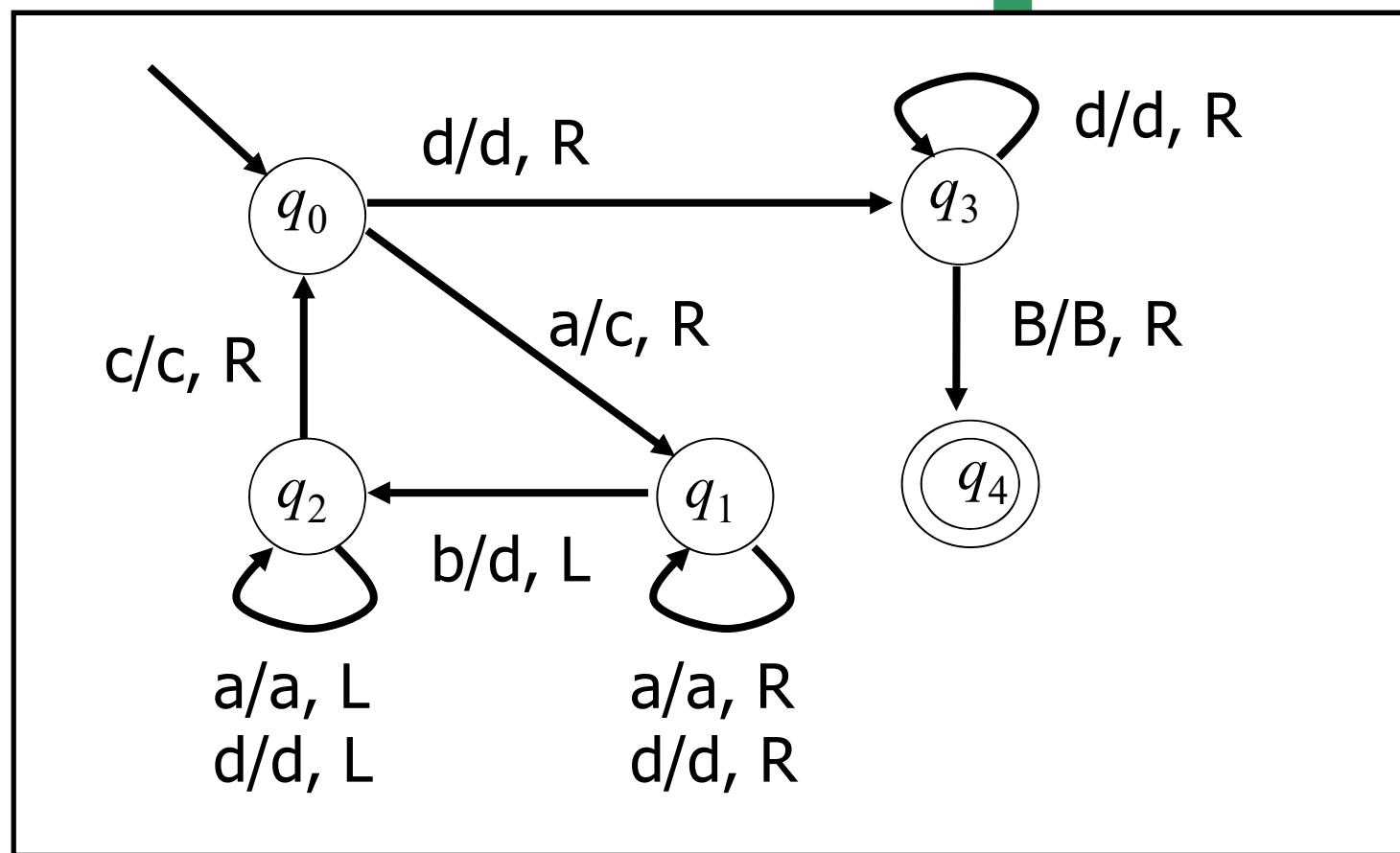
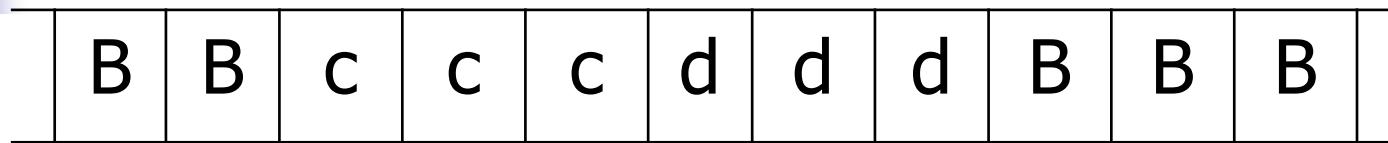
$\{a^i b^i \mid i \in \mathbb{N}\}$ を受理するTM (23)



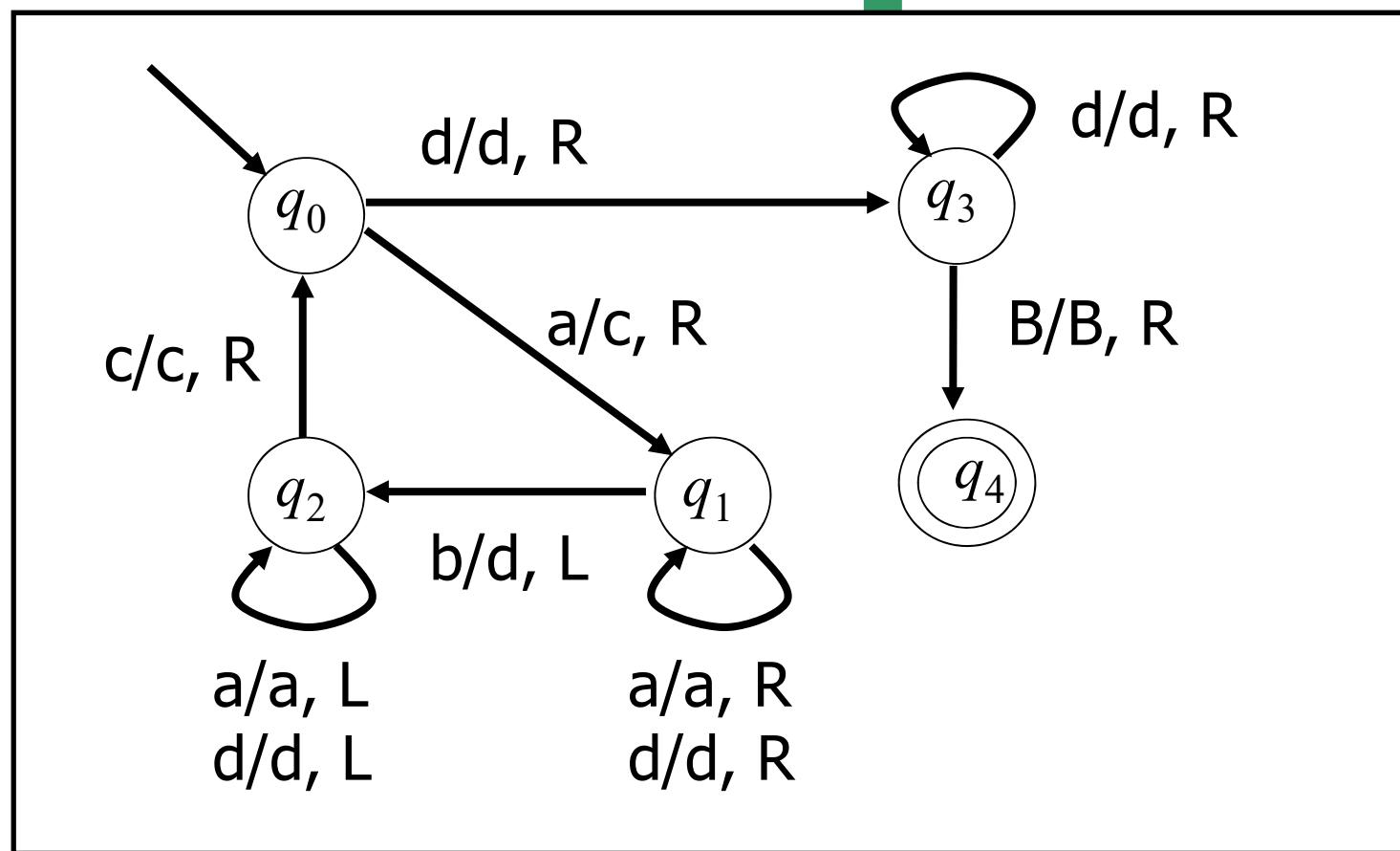
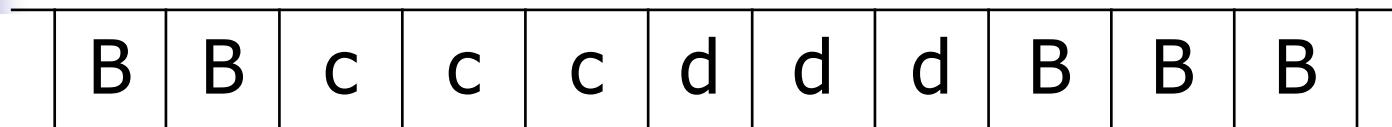
$\{a^i b^i \mid i \in \mathbb{N}\}$ を受理するTM (24)

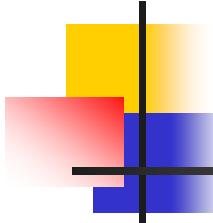


$\{a^i b^i \mid i \in \mathbb{N}\}$ を受理するTM (25)



$\{a^i b^i \mid i \in \mathbb{N}\}$ を受理するTM (26)





TMが受理する言語

Turing機械 $M = (\Gamma, \Sigma, S, \delta, s_0, B, F)$ に対して

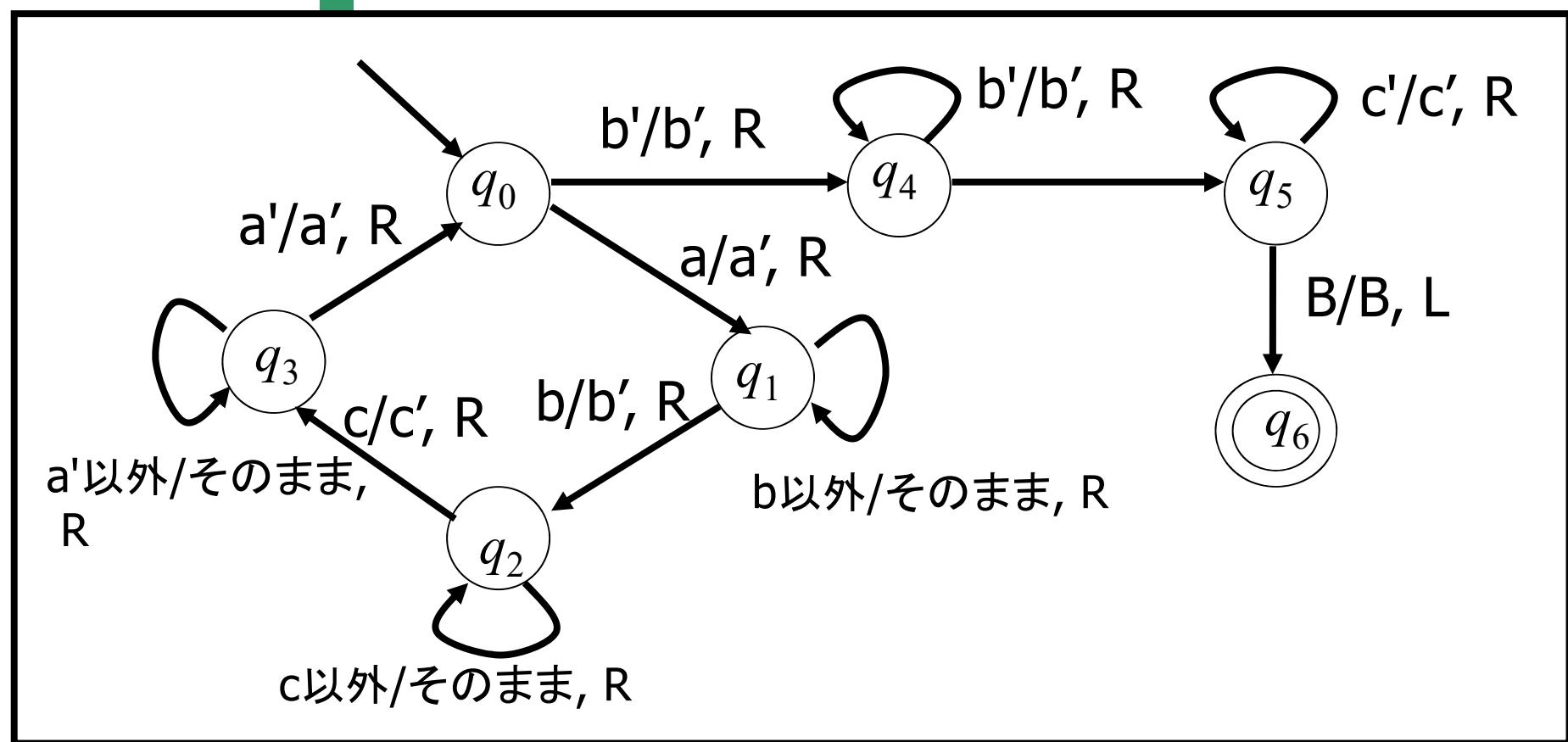
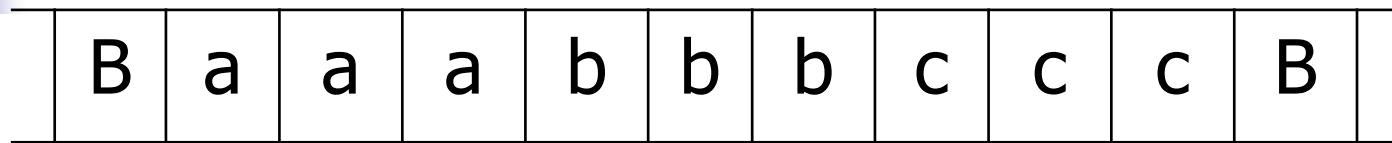
- 語 $w = c_1 \dots c_k$ が M に受理される

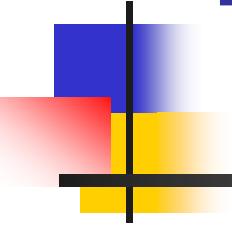
\Leftrightarrow 動作列 $(q_0, c_1 \dots c_k, 1) \Rightarrow \dots \Rightarrow (q_h, c_1 \dots c_i \dots c_k, i)$ で

$(q_h, c_1 \dots c_k, j)$ は終端計算状況かつ $q_h \in F$ を満たすものを構成することができる

- $L(M) = \{ w \in \Sigma^* \mid w \text{ が } M \text{ に受理される} \}$

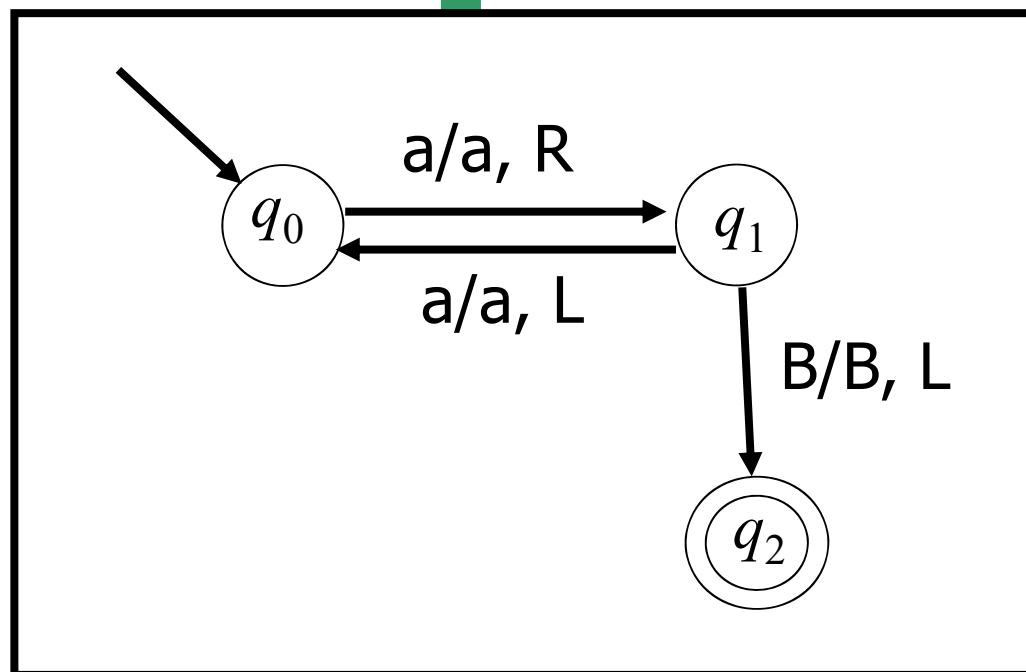
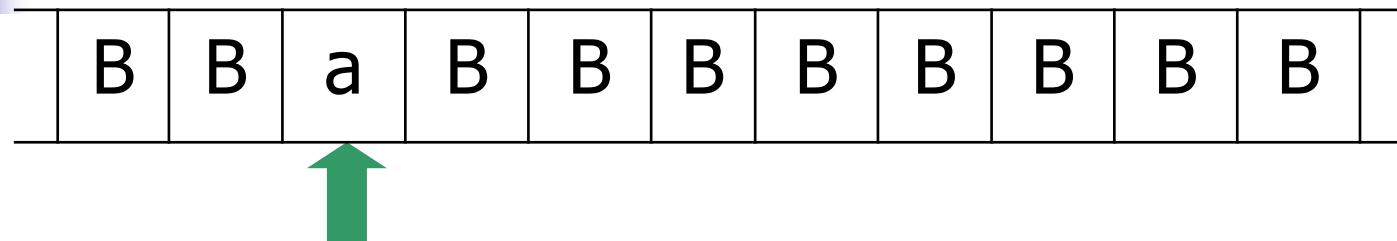
$\{a^i b^i c^i \mid i \in \mathbb{N}\}$ を受理するTM (1)





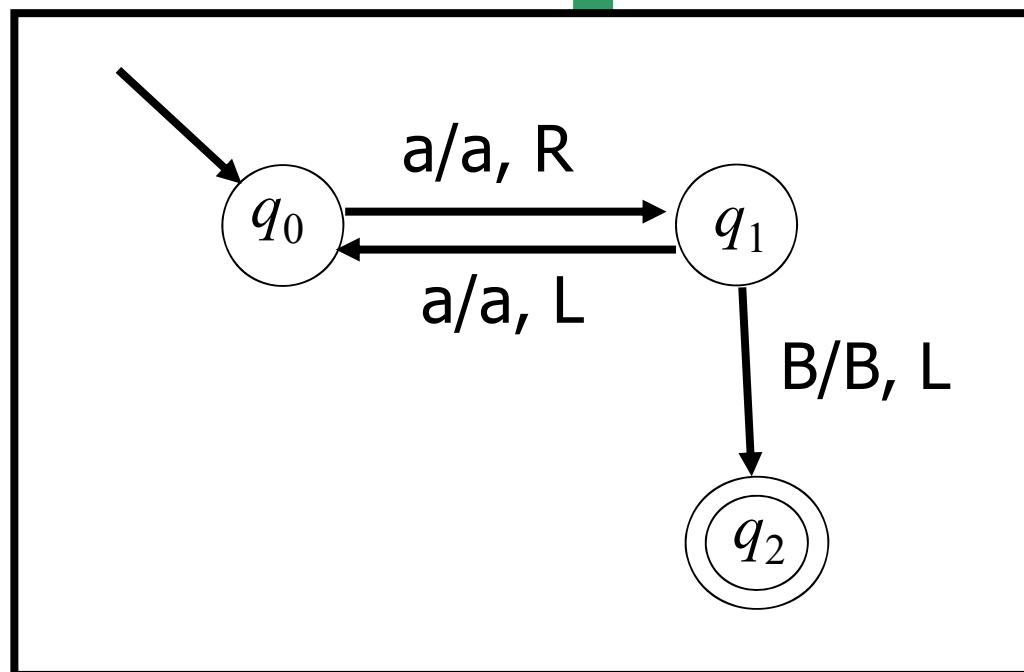
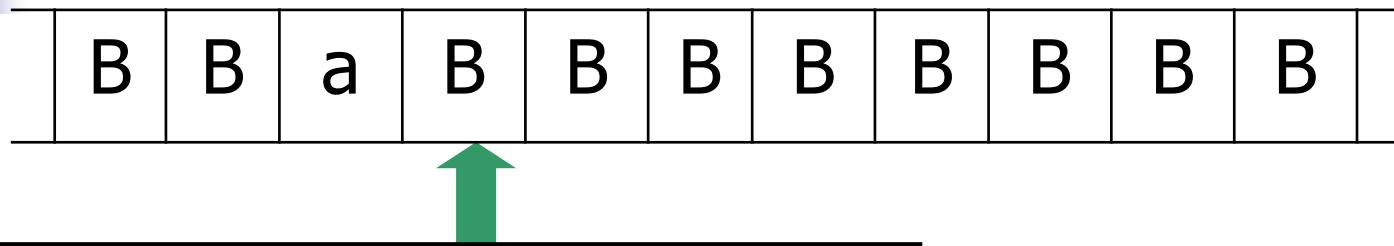
Turing機械の停止性

入力によっては停止しないTM(1-1)



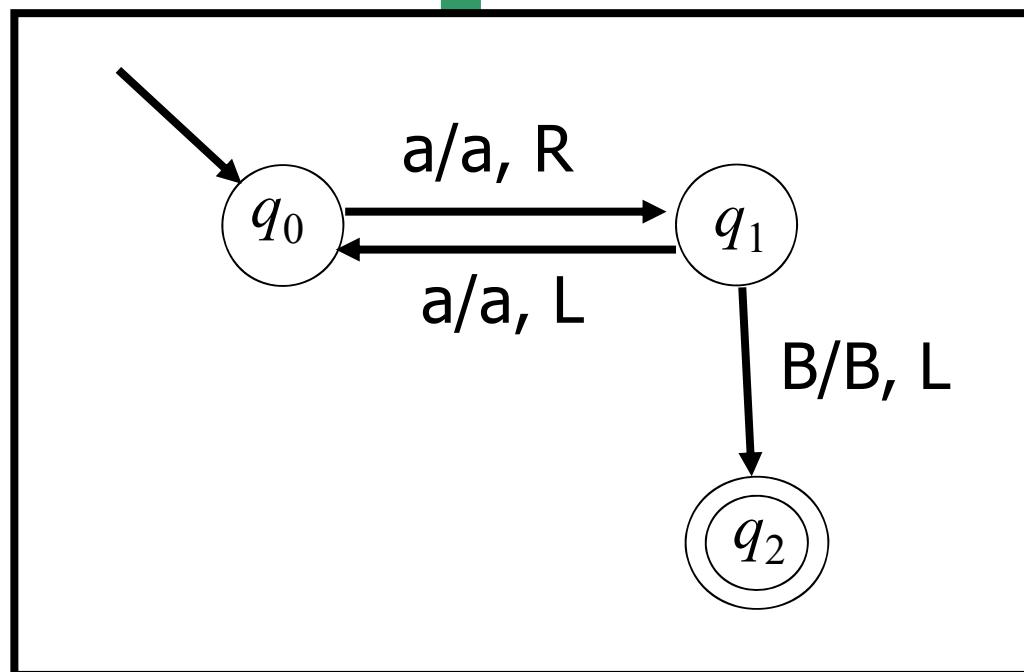
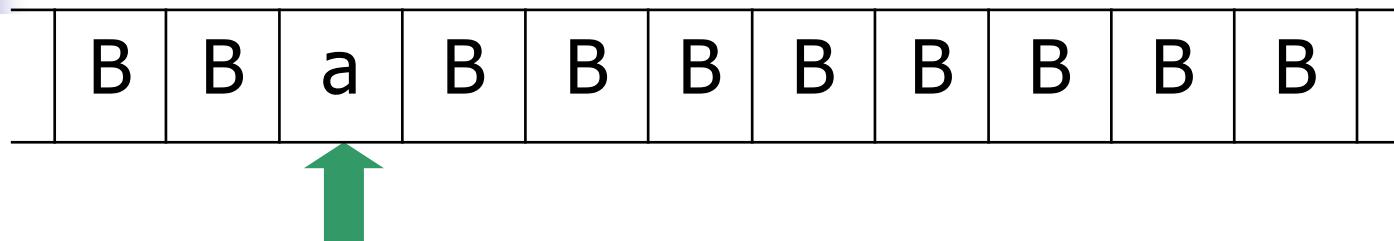
- 語 a は M に受理される
- 語 a^n ($n \geq 2$) に対しては動作(遷移)が停止しない
 - この場合も“受理しない”

入力によっては停止しないTM(1-2)



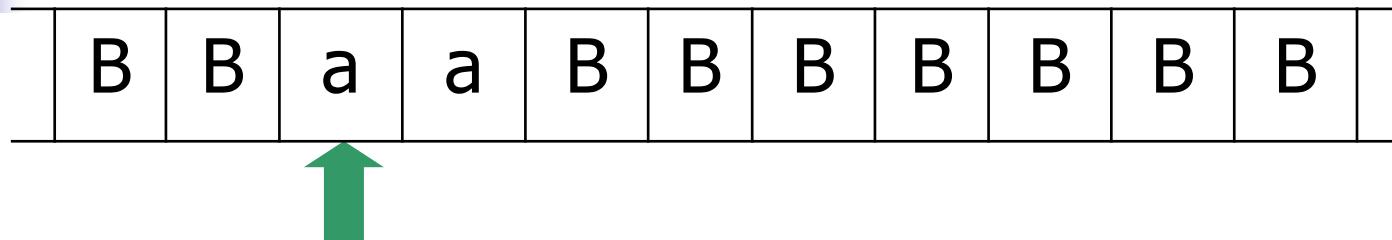
- 語 a は M に受理される
- 語 a^n ($n \geq 2$) に対しては動作(遷移)が停止しない
 - この場合も“受理しない”

入力によっては停止しないTM(1-3)



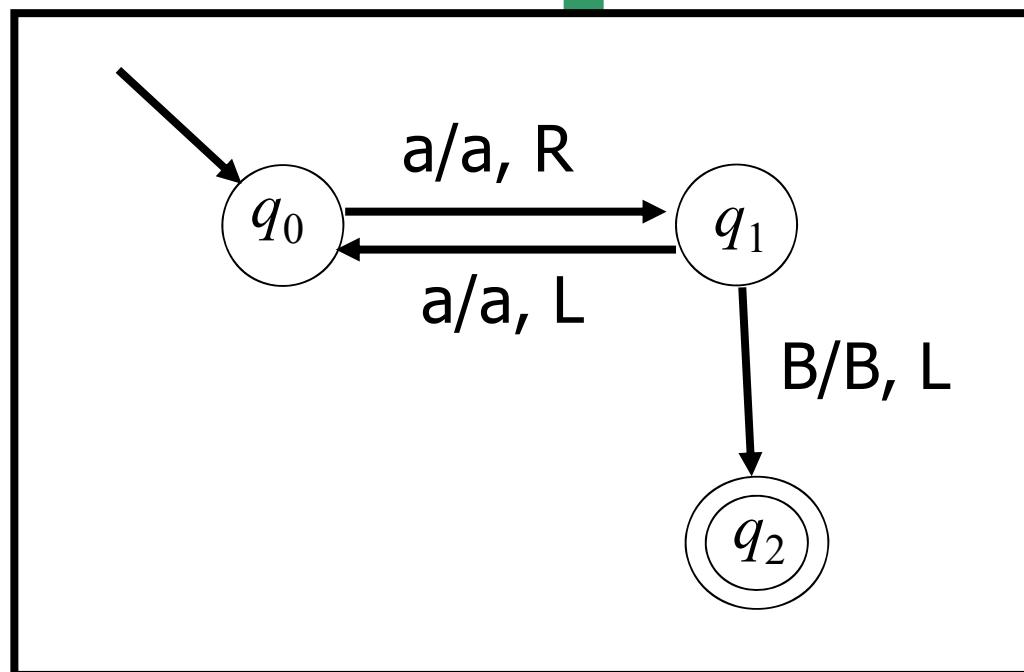
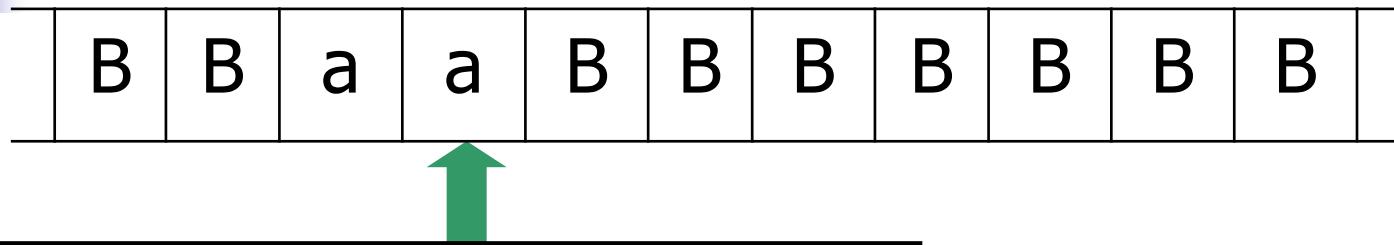
- 語 a は M に受理される
- 語 a^n ($n \geq 2$) に対しては動作(遷移)が停止しない
 - この場合も“受理しない”

入力によっては停止しないTM(2-1)



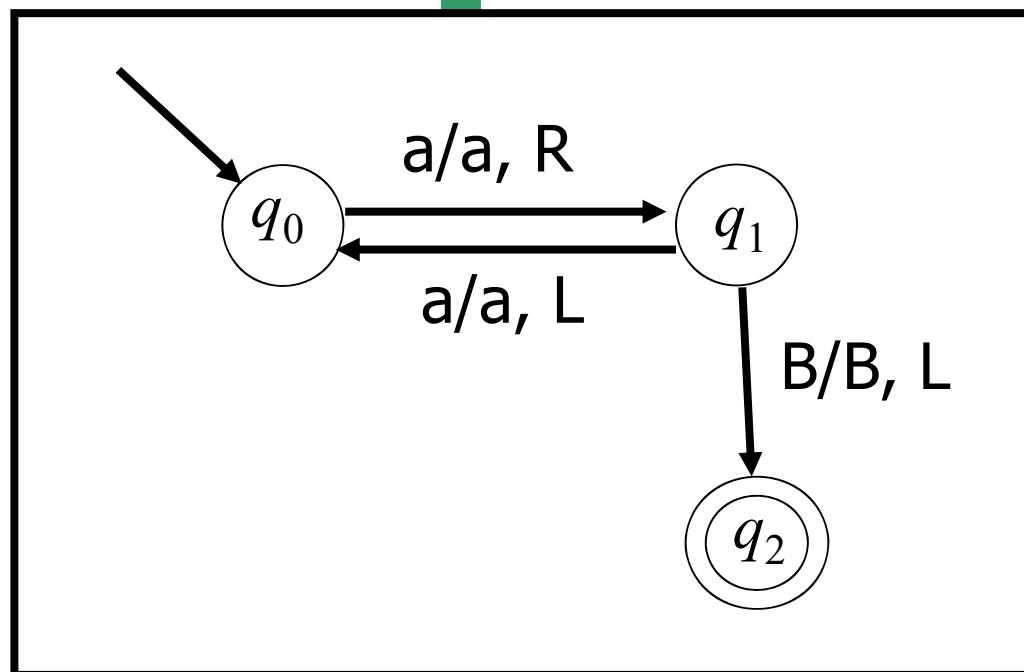
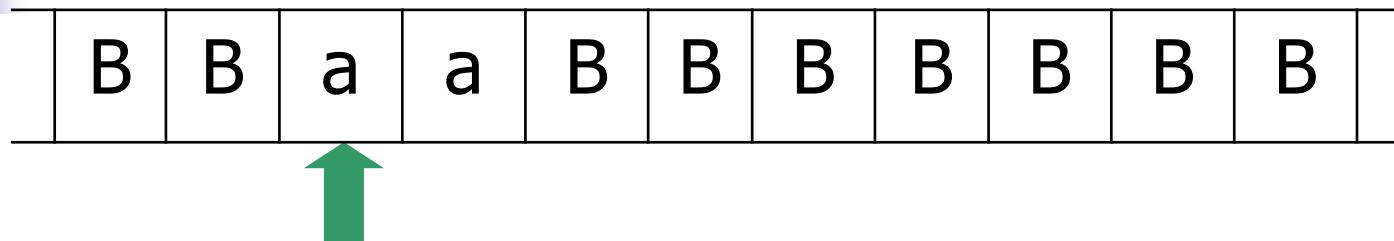
- 語 a は M に受理される
- 語 a^n ($n \geq 2$) に対しては動作(遷移)が停止しない
 - この場合も“受理しない”

入力によっては停止しないTM(2-2)

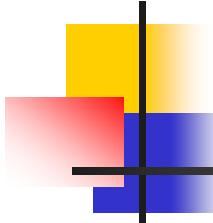


- 語 a は M に受理される
- 語 a^n ($n \geq 2$) に対しては動作(遷移)が停止しない
 - この場合も“受理しない”

入力によっては停止しないTM(2-3)



- 語 a は M に受理される
- 語 a^n ($n \geq 2$) に対しては動作(遷移)が停止しない
 - この場合も“受理しない”

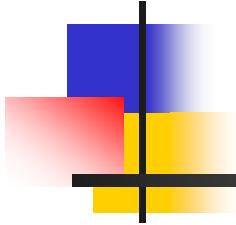


TMの停止性は決定不能

- 受理機械としてのTuring機械 M が**任意の語** w に対して停止するかどうかを決定するアルゴリズムは構成することができない

帰納的可算集合: 受理するTMが構成できる.

帰納的集合: 受理するTMで,必ず停止する.



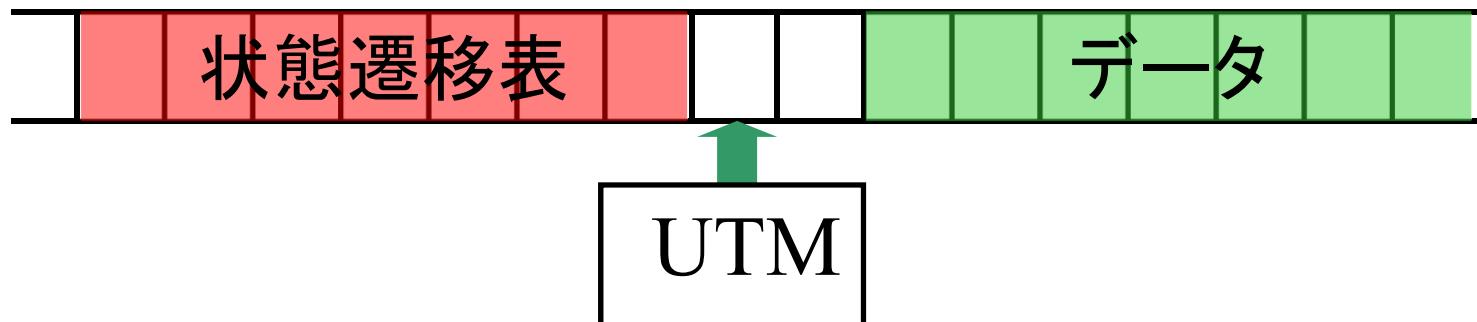
万能Turing機械

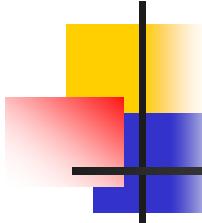
Universal Turing Machine

- 状態遷移図は状態遷移表として表現できるのだから、(記号をうまく設定すれば)テープに記録することができる。



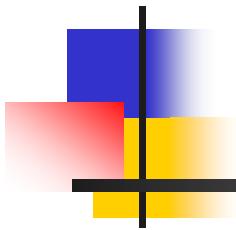
- 状態遷移表(**プログラム**)を参照しながら、次の遷移を決定するようなTM(**UTM**)を構成すれば、どのようなTMも模倣できる。



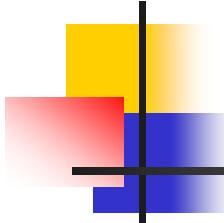


UTMの帰結

- 「自分が構成したTMは停止するのか？」を一般的に判定するようなTMは存在しない (Turing)
⇒ 決定不可能な問題の存在
 - Entscheidungsproblem : 決定問題
- 「プログラムをメモリー上に置く」という方式の計算機の開発
 - しかし、この方式はコンピュータ・ウィルスの出現を許容してしまう。

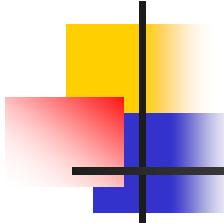


句構造文法



句構造文法

- 生成規則 $\alpha \rightarrow \beta$ の α, β は
 - 非終端記号(構文要素)
 - 文, 名詞句, 動詞句, 代名詞, 動詞, ...
 - 終端記号(単語・記号)
- の有限列
- 文法 G : 生成規則の集合
 - と非終端記号, 終端記号, 文全体(開始)を表す非終端記号
- G によって生成される文: “文” に生成規則を有限回適用して得られる終端記号列



句構造文法

- 句構造文法 $G = (N, \Sigma, P, S)$

N : 非終端記号(構文要素)の有限集合

Σ : 終端記号(単語・記号)の有限集合

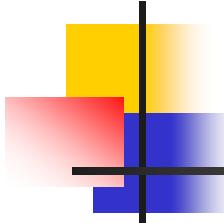
P : 生成規則の有限集合

$S \in N$: 開始記号

- **生成規則:** $\alpha \rightarrow \beta$ という形の規則

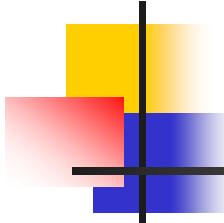
$\alpha \in (N \cup \Sigma)^+, \beta \in (N \cup \Sigma)^*$

- α, β は非終端記号(構文要素)と終端記号(単語・記号)の有限列



句構造文法の例(1)

$G_4 = (N = \{S, A, B, C, T\}, \Sigma = \{a, b\},$
 $P = \{S \rightarrow aBCT, S \rightarrow aBC,$
 $T \rightarrow ABCT, T \rightarrow ABC,$
 $BA \rightarrow AB, CA \rightarrow AC, CB \rightarrow BC,$
 $aA \rightarrow aa, aB \rightarrow ab, bB \rightarrow bb, bC \rightarrow bc,$
 $cC \rightarrow cc\}, S)$

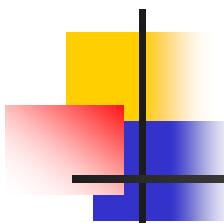


句構造文法の例(2)

$G_1 = (N = \{S\}, \Sigma = \{a, b\}, P = \{S \rightarrow ab, S \rightarrow aSb\}, S)$

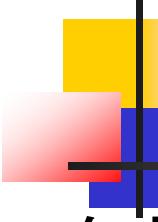
$G_2 = (N = \{S, A, B\}, \Sigma = \{a, b\},$
 $P = \{S \rightarrow aAB, A \rightarrow bBB, B \rightarrow abb\}, S)$

$G_3 = (N = \{S, A, B, C\}, \Sigma = \{a, b\},$
 $P = \{S \rightarrow aA, S \rightarrow bB, A \rightarrow aA, A \rightarrow bC,$
 $A \rightarrow b$
 $B \rightarrow aB, B \rightarrow bB, C \rightarrow aA, C \rightarrow bC,$
 $C \rightarrow b\}, S)$



句構造文法の例(3)

$$G_5 = (N = \{\text{式}\}, \Sigma = \{a, b, c, +, *, (,)\},$$
$$P = \{\text{式} \rightarrow (\text{式} + \text{式}), \text{式} \rightarrow (\text{式} * \text{式})$$
$$\text{式} \rightarrow a, b, c \}, \text{式})$$



句構造文法による導出

- 句構造文法 $G = (N, \Sigma, P, S)$

列 $\gamma\alpha\delta$ ($\gamma, \alpha, \delta \in (N \cup \Sigma)^*$) に対して生成規則 $\alpha \rightarrow \beta \in P$ が見つかれば、 $\gamma\alpha\delta \Rightarrow \gamma\beta\delta$ とかく。

- 列 $\gamma\beta\delta$ は $\gamma\alpha\delta$ から $\alpha \rightarrow \beta$ によって **直接に導出される** という。
- 一つの列に α が 2 箇所以上みつかっても、それらを **同時に** β に 書換えることはできない。

$$S * (S + S) \Rightarrow V * (S + S)$$

$$S * (S + S) \Rightarrow S * (V + S)$$

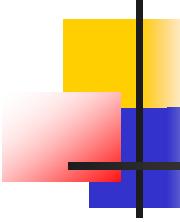
$$S * (S + S) \not\Rightarrow V * (V + V)$$

- 列 α と β は、 $\alpha = \beta$ であるか、または、

$$\alpha = \gamma_1 \Rightarrow \gamma_2 \Rightarrow \dots \Rightarrow \gamma_n = \beta$$

を満たす $\gamma_1, \gamma_2, \dots, \gamma_n$ が構成できるとき $\alpha \Rightarrow^* \beta$ とかく

- 列 β は α から G によって導出されるという。



例(2)

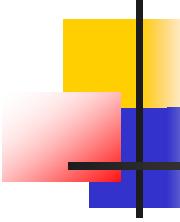
$G = (N = \{S, A, B, C\}, \Sigma = \{\text{a, b}\},$
 $P = \{S \rightarrow \text{a}A, S \rightarrow \text{b}B, A \rightarrow \text{a}A, A \rightarrow \text{b}C, A \rightarrow \text{b}$
 $B \rightarrow \text{a}B, B \rightarrow \text{b}B, C \rightarrow \text{a}A, C \rightarrow \text{b}C, C \rightarrow \text{b}\},$
 $S)$

$S \Rightarrow \text{a}A \Rightarrow \text{aa}A \Rightarrow \text{aab}C \Rightarrow \text{aab}$

$S \Rightarrow \text{a}A \Rightarrow \text{aa}A \Rightarrow \text{aab}C \Rightarrow \text{aabb}$

$S \Rightarrow \text{a}A \Rightarrow \text{ab}C \Rightarrow \text{aba}A \Rightarrow \text{abab}$

$S \Rightarrow \text{a}A \Rightarrow \text{ab}C \Rightarrow \text{abb}AC \Rightarrow \text{abba}A \Rightarrow \text{abbaa}A$
 $\Rightarrow \text{abbaaa}A \Rightarrow \text{abbaaaab}$

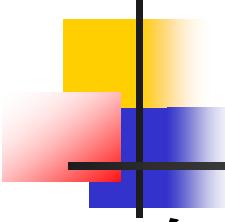


例(3)

$G_4 = (N = \{S, A, B, C, T\}, \Sigma = \{\mathbf{a}, \mathbf{b}\},$
 $P = \{S \rightarrow \mathbf{a}BCT, S \rightarrow \mathbf{a}BC,$
 $T \rightarrow ABCT, T \rightarrow ABC,$
 $BA \rightarrow AB, CA \rightarrow AC, CB \rightarrow BC,$
 $\mathbf{a}A \rightarrow \mathbf{aa}, \mathbf{a}B \rightarrow \mathbf{ab}, \mathbf{b}B \rightarrow \mathbf{bb}, \mathbf{b}C \rightarrow \mathbf{bc},$
 $\mathbf{c}C \rightarrow \mathbf{cc}\}, S)$

$S \Rightarrow \mathbf{a}BC \Rightarrow \mathbf{ab}C \Rightarrow \mathbf{abc}$

$S \Rightarrow \mathbf{a}BCT \Rightarrow \mathbf{a}BCABC \Rightarrow \mathbf{a}BACBC \Rightarrow \mathbf{a}BABCC$
 $\Rightarrow \mathbf{a}ABBCC \Rightarrow \mathbf{aa}BBCC \Rightarrow \mathbf{aab}BCC \Rightarrow \mathbf{aabb}CC$
 $\Rightarrow \mathbf{aabbc}C \Rightarrow \mathbf{aabbcc}$



句構造文法が生成する言語

- 句構造文法 $G = (N, \Sigma, P, S)$ に対して

$$L(G) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}$$

を G が **生成する言語** という

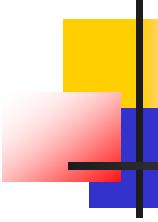
- 言語 L は $L = L(G)$ なる句構造文法が存在すると
きに **帰納的に可算**(recursively enumerable) であ
るという

(注) 数学でいう“可算集合”とは、自然数全体の集
合 \mathbb{N} と 1 対 1 対応がある集合のことである。

形式言語はすべて可算集合であるが、帰納的
に可算な言語とは限らない。

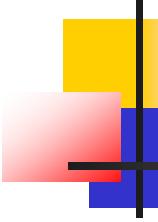


句構造文法とTM



集合と関数によるTMの表現

- 以下のような構成要素からなる組 $M = (\Gamma, \Sigma, S, \delta, s_0, B, F)$ を(決定性)Turing機械とよぶ.
 - Γ はアルファベット: テープ記号の集合
 - $\Sigma \subset \Gamma$ はアルファベット: 入力記号
 - S は空でない有限集合であり,
その要素を状態とよぶ.
 - δ は $S \times \Gamma \rightarrow S \times \Gamma \times \{L, R\}$ なる関数
 - $s_0 \in S$ は特定の状態であり, 初期状態とよぶ.
 - $B \in \Gamma - \Sigma$ は特定の記号であり, 空白記号とよぶ
 - $F \subset S$ は特定の状態の集合であり, その要素を終了状態, もしくは受理状態とよぶ.



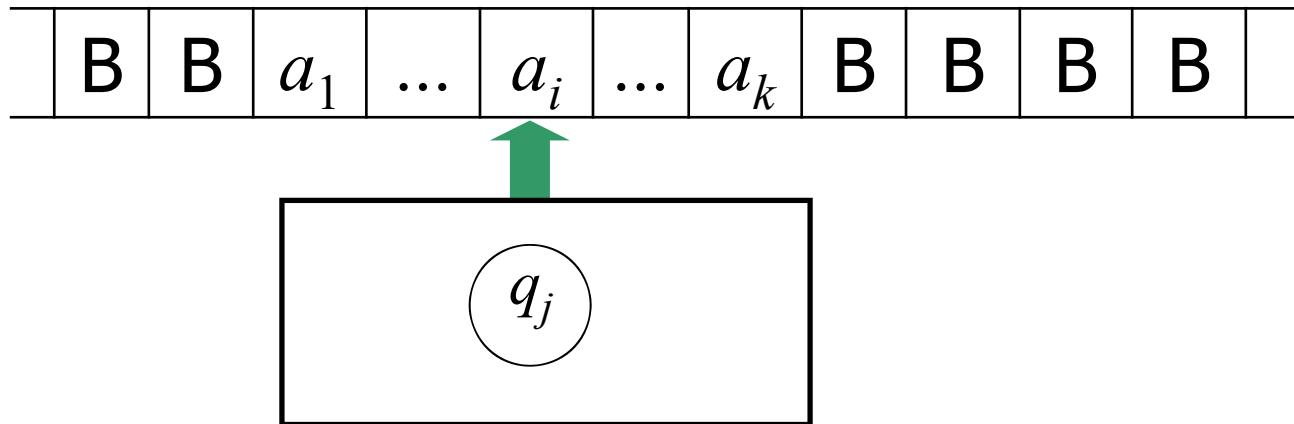
状態遷移表による表現

- テープ記号: B, c_1, \dots, c_n
- 状態: $q_0, q_1, q_2, \dots, q_m$
- ヘッドを動かす方向: $D = L$ or R
- 遷移関数 $\delta(q_i, c_j) = \langle q_k, c_h, D \rangle$
 - 引数の組 (q_i, c_j) によっては値が定義されない
⇒ 関数としては特別な値 λ (あるいは \perp) を返すとする
機械としては特別な状態に遷移すると解釈する

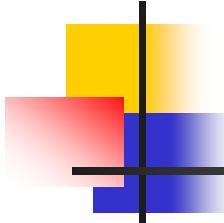
	F	B	c_1	...	c_n
q_0					
q_1					
...		
q_m					

計算状況(時点表示)

- 無限長のテープの有限部分のみ空白でない記号が記入されている $a_1 \dots q_j a_i \dots a_k$
- 時点表示 $\sigma = (q_j, a_1 \dots a_i \dots a_k, i)$ または $a_1 \dots q_j a_i \dots a_k$

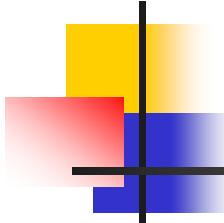


- $(q_j, a_1 \dots a_i \dots a_k, i) \Rightarrow (q_k, a_1 \dots a_l \dots a_k, l)$
遷移表を1回だけ用いて $\sigma = (q_j, a_1 \dots a_i \dots a_k, i)$ が
 $\tau = (q_h, a_1 \dots a_l \dots a_k, l)$ ($l = \pm i$) に遷移する



終端計算状況

- 計算状況 $(q_h, c_1 \dots c_i \dots c_k, i)$ が終端計算状況
 $\Leftrightarrow \delta(q_h, c_i)$ が定義されていない

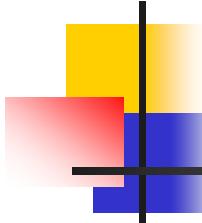


句構造文法からTMへの変換

- 句構造文法 $G = (N, \Sigma, P, S)$ に対して
 $\Gamma \supset N \cup \Sigma \cup \{B, \#\}$ として TM $M = (\Gamma, \Sigma, Q, \delta, s_0, B, F)$ を構成する

構成の基本的アイデア

- テープ上には入力記号列 w と S からの導出の途中経過 $\sigma \in (N \cup \Sigma)^*$ を $\# \sigma \# w \#$ の形で保持しておく
 - 計算の開始時点では $\# S \# w \#$ がテープ上にある
- 生成規則 $\alpha \rightarrow \beta$ に対して, σ の中の部分列 α を非決定的に探索し, それを β に書き換える
- 得られた σ が w に一致しているかどうかを確認する



TMから句構造文法への変換

- 非終端記号の集合:

$$N = Q \cup \{S, S_1, S_2\} \cup \{[ac] \mid a \in \Sigma, c \in \Gamma\} \cup \{[a] \mid a \in \Sigma\}$$

- 入力記号列 $a_1a_2\dots a_n$ を受理するときの終端計算状況が $(q_h, c_1\dots c_i\dots c_k, i)$ のとき

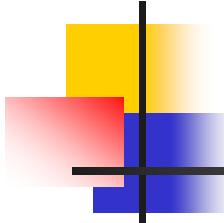
$$S \Rightarrow^* q_0[a_1a_1] [a_2a_2]\dots [a_na_n] [\mathbf{B}]^k \quad (1)$$

$$\Rightarrow^* [a_1c_1]\dots [a_{i-1}c_{i-1}] q_h [a_ic_i][a_{i+1}c_{i+1}]\dots [a_{i+k}c_{i+k}] \quad (2)$$

$$\Rightarrow^* a_1a_2\dots a_n \quad (3)$$

ただし k は非決定的に選ぶ

(2)の部分で $a_{n+1}=a_{n+1}=\dots=\varepsilon$



句構造文法の構成

(1)の部分

$$S \rightarrow q_0 S_1, S_1 \rightarrow [aa]S_1 \quad (a \in \Sigma), S_1 \rightarrow S_2, S_2 \rightarrow [\mathbf{B}]S_2, S_2 \rightarrow \varepsilon$$

(2)の部分

$$\delta(q_i, c_j) = \langle q_k, c_h, R \rangle \text{ のとき}$$

$$q_i[ac_j] \rightarrow [ac_h]q_k \quad (a \in \Sigma \cup \{\varepsilon\})$$

$$\delta(q_i, c_j) = \langle q_k, c_h, L \rangle \text{ のとき}$$

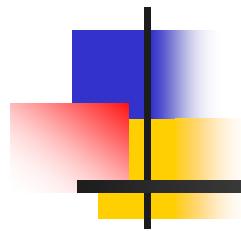
$$[ad]q_i[ac_j] \rightarrow q_k[ad][ac_h] \quad (a \in \Sigma \cup \{\varepsilon\}, d \in \Gamma)$$

(3)の部分

$$q_h \in F \text{ のとき } [ad]q_h \rightarrow q_h a q_h, q_h[ad] \rightarrow q_h a q_h, q_h \rightarrow \varepsilon$$

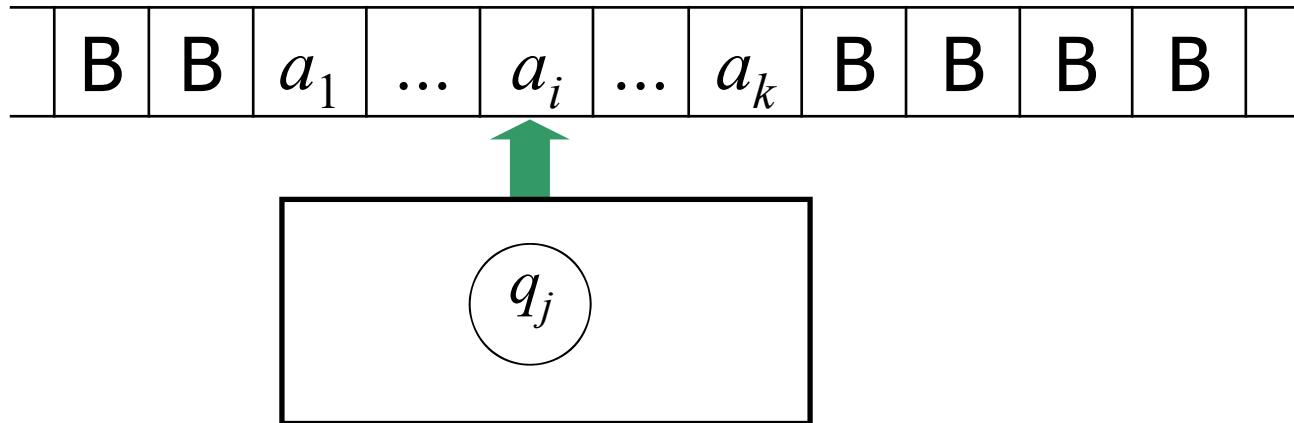
$$(a \in \Sigma \cup \{\varepsilon\}, d \in \Gamma)$$

片側テープTM, 多テープTM, 非決定性TM



計算状況(時点表示)

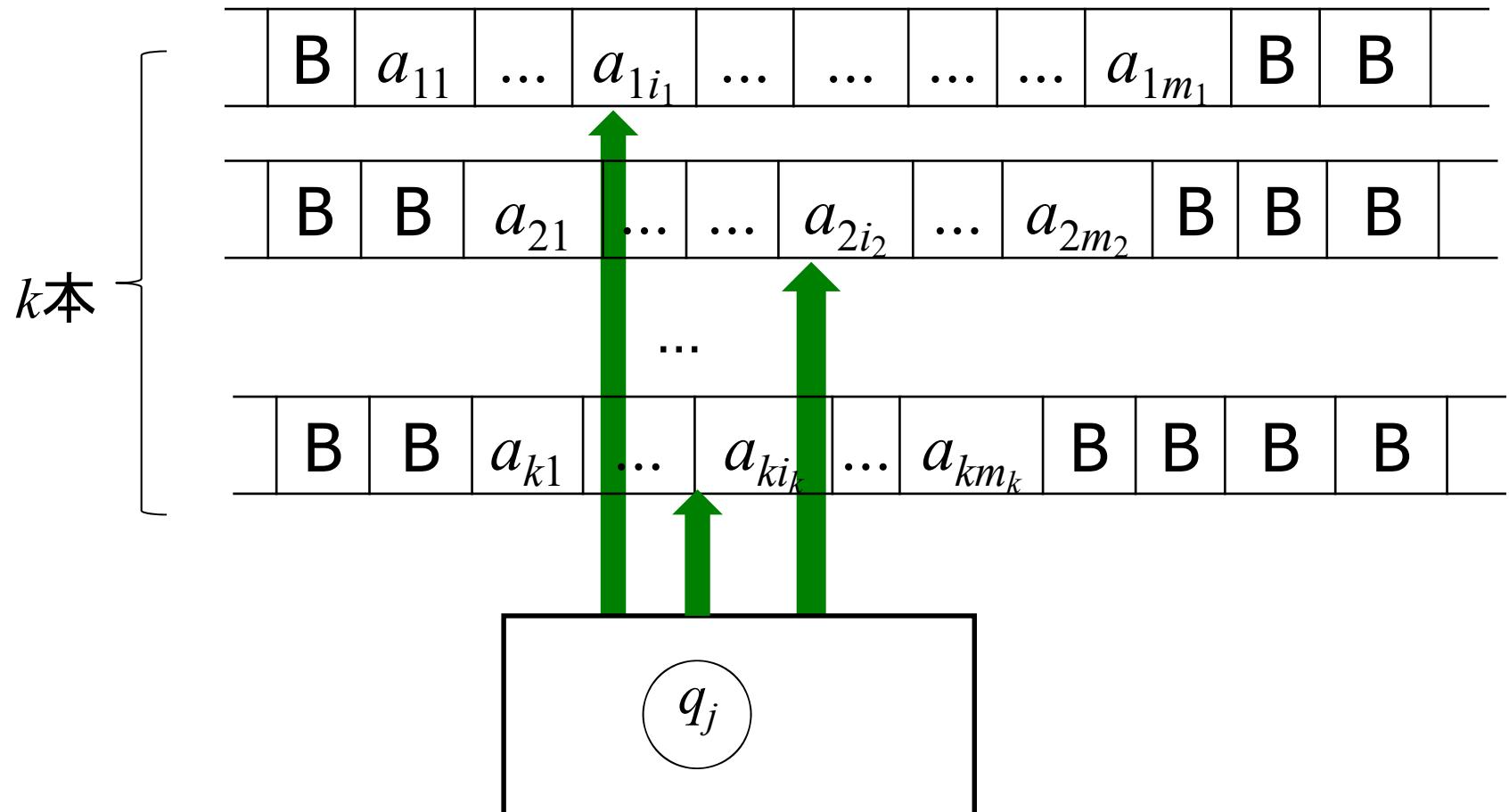
- 無限長のテープの有限部分のみ空白でない記号が記入されている $a_1 \dots q_j a_i \dots a_k$
- 時点表示 $\sigma = (q_j, a_1 \dots a_i \dots a_k, i)$ または $a_1 \dots q_j a_i \dots a_k$

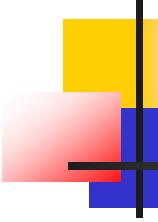


- $(q_j, a_1 \dots a_i \dots a_k, i) \Rightarrow (q_k, a_1 \dots a_l \dots a_k, l)$
遷移表を1回だけ用いて $\sigma = (q_j, a_1 \dots a_i \dots a_k, i)$ が
 $\tau = (q_h, a_1 \dots a_l \dots a_k, l)$ ($l = \pm i$) に遷移する

多テーブTM

- 多テーブTM: テープを k 本持つ:

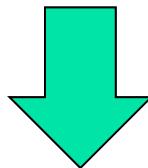




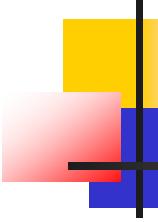
多テーブルTMと1テーブルTM(1)

- 遷移関数 $\delta(q_i, \langle c_1, \dots, c_k \rangle) = \langle q_k, \langle d_1, \dots, d_k \rangle, \langle D_1, \dots, D_k \rangle \rangle$
- 時点表示 $\sigma =$

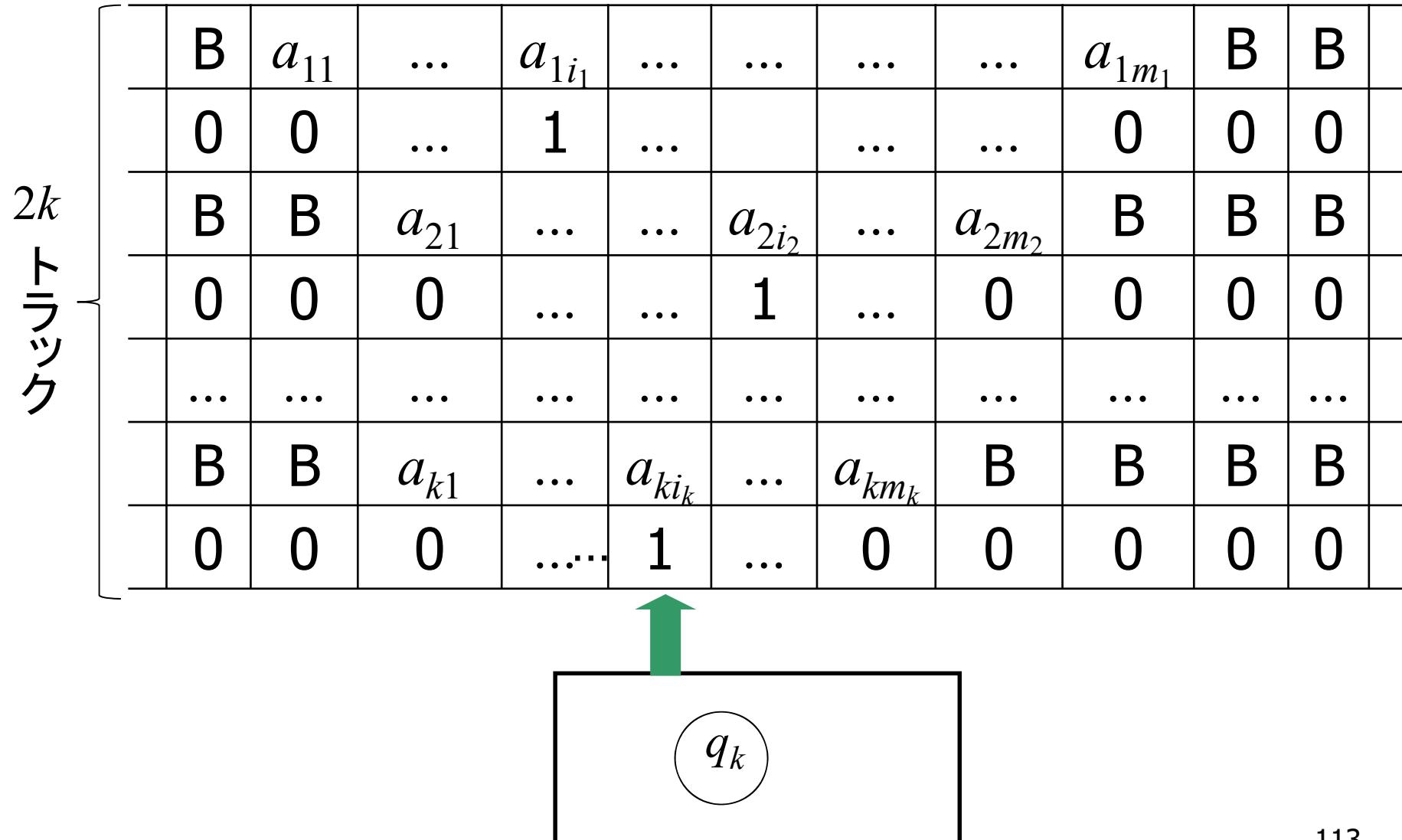
$(q_j, (a_{11}..a_{1i_1}..a_{1m_1}, i_1), (a_{21}..a_{2i_2}..a_{2m_2}, i_2), \dots, (a_{k1}..a_{ki_k}..a_{m_k}, i_k))$



- 記号 $c_i \in \Gamma$ と ヘッドマーク の有無を表す bit b_i の組のベクトル $\langle \langle c_1, b_1 \rangle, \dots, \langle c_k, b_k \rangle \rangle$ を 記号とみなすことにより、 多テーブルTM M を 1テーブルTM を構成することができる

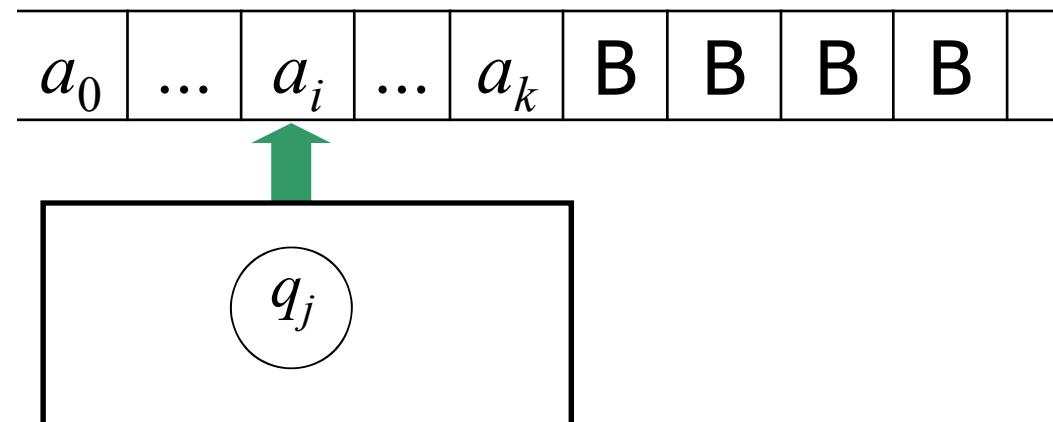


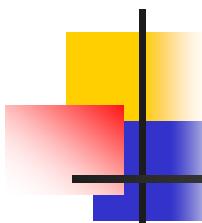
多テーブルTMと1テーブルTM(1)



片側テープTM

- TMにおいて“テープが両方向に無限に伸びている”を“片側だけに無限に伸びている”にしたもの



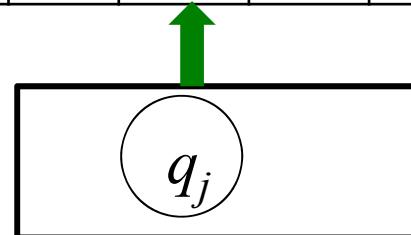


両側テープTMの模倣(1)

- 両側テープTM M のセルのアドレスを**整数** $i \in \mathbb{Z}$ で与える
- 片側テープTM M' のセルのアドレスを**非負整数** $j \in \mathbb{N}$ で与える
- 記号 $c_i \in \Gamma$ とヘッドマーカの有無を表すbit b_i の組のベクトル $\langle\langle c_+, b_+ \rangle, \langle c_-, b_- \rangle \rangle$ を記号とみなすことで、両側テープTM M を模倣する片側テープTMを構成することができる。

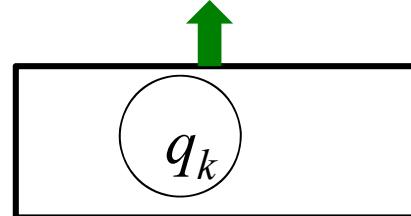
両側テープTMの模倣(2)

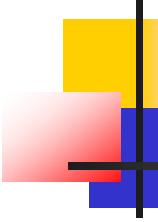
	...	n	...	0	1	...	i	...	k	
	B	a_{-n}	...	a_0	a_1	...	a_i	...	a_k	B	B	



4トラック

0	1	...	i	...	n	...	k	...	
a_0	a_1	...	a_i	...	a_n	...	a_n	B	
0	0		1		0		0	0	
B	a_{-1}	...	a_{-i}	...	a_{-n}	...	B	B	
0	0	...	0	0	



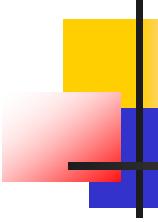


非決定性TMの言語受理能力(1)

- 非決定性TMを模倣する決定性TMを構成できる
(注) 言語の受理能力という観点からの主張であって、計算量の観点を無視している!

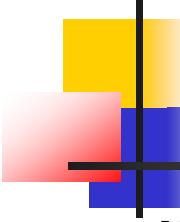
証明のアイデア:

- 各状態に対する非決定の選択肢の数には上限 M がある
- 各選択肢に0から $M-1$ までの数字を振っておく
- 計算過程(状態遷移)で選んだ選択肢を並べると $\{0,1,\dots,M-1\}^*$ の要素になっているはずである



非決定性TMの言語受理能力(2)

- 3テープTMを用意する。
- 第1テープには入力記号を保存する
- 第2テープには $\{0,1,\dots,M-1\}^*$ の要素を次々と発生させる
- 第2テープにある $\{0,1,\dots,M-1\}^*$ の要素を“選択の指示”と解釈して、第3テープを用いて計算を行う



TMが受理する言語(1)

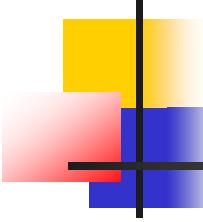
- 帰納的に可算な言語を

$L=L(M)$ なるTM M が存在する言語
としてもよい

- TMを(受理機械ではなく)計算機械とみなしたとき,帰納的に可算な言語とは, 非負整数 $n \in \mathbf{N}^+$ を入力とし, Σ^* の要素を出力とするTM $M(n)$ で

$$L=\{ w=M(n) \mid n \in \mathbf{N}^+ \}$$

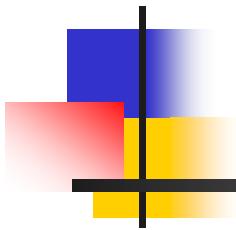
と表される言語と定義してもよい.



TMが受理する言語(2)

- TMは語 w を受理するとき停止するが, 受理しないとき停止するとは限らない
- 語 w を受理するときもしないときも必ず停止するようなTM M が存在するとき, $L=L(M)$ は**帰納的**(recursive)であるという

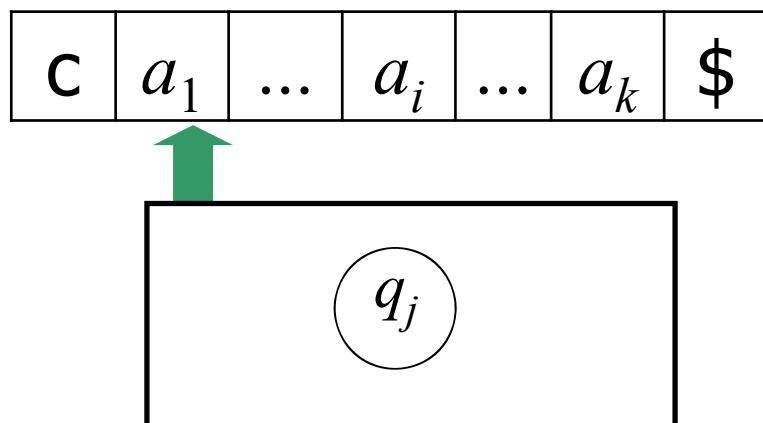
(注) 帰納的可算な言語が帰納的であるとは限らない.

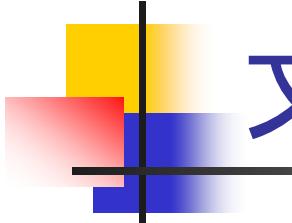


線形拘束オートマトン

線形拘束オートマトン

- $\Gamma \subset \{c, \$\}$ とし、入力記号列の左側に、右側に \$ が置かれ、ヘッドは c よりも左、\$ よりも右に動くことができないような TM を **線形拘束(有界)オートマトン** (Linear Bounded Automaton) という





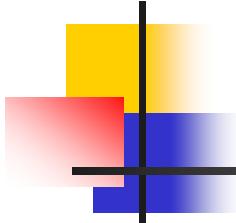
文脈依存言語とLBA

- 文脈依存文法 G で生成される言語 $L(G)$ を受理する LBA を構成することができる。
- LBA M で受理される言語 $L(M)$ は文脈依存言語である



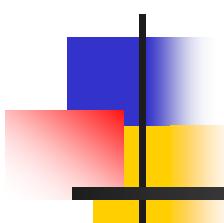
LBAとTM

- 文脈依存言語は帰納的である.
- 帰納的であるが文脈依存でない言語が存在する.

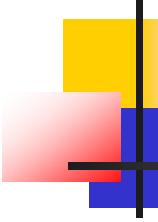


言語・文法・受理機械の階層

言語族	形式文法のクラス	受理機械の種類
帰納的可算	句構造文法	TM
帰納的		停止するTM
文脈依存	文脈依存文法	LBA
文脈自由	文脈自由文法	NPDA
決定性文脈 自由	LR(k)文法 ($k \geq 1$)	DPDA
正則	正則文法	FA

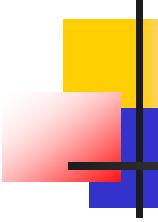


TMと句構造文法



集合と関数によるTMの表現

- 以下のような構成要素からなる組 $M = (\Gamma, \Sigma, S, \delta, s_0, B, F)$ を(決定性)Turing機械とよぶ.
 - Γ はアルファベット: テープ記号の集合
 - $\Sigma \subset \Gamma$ はアルファベット: 入力記号
 - S は空でない有限集合であり,
その要素を状態とよぶ.
 - δ は $S \times \Gamma \rightarrow S \times \Gamma \times \{L, R\}$ なる関数
 - $s_0 \in S$ は特定の状態であり, 初期状態とよぶ.
 - $B \in \Gamma - \Sigma$ は特定の記号であり, 空白記号とよぶ
 - $F \subset S$ は特定の状態の集合であり, その要素を終了状態, もしくは受理状態とよぶ.

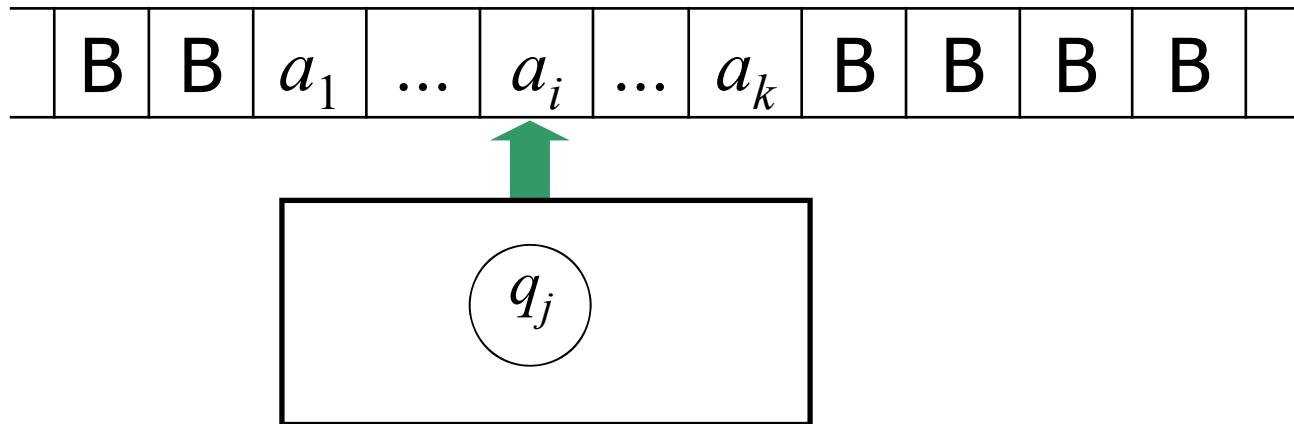


集合と関数によるTMの表現

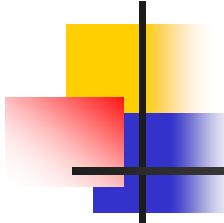
- 以下のような構成要素からなる組 $M = (\Gamma, \Sigma, S, \delta, s_0, B, F)$ を(決定性)Turing機械とよぶ.
 - Γ はアルファベット: テープ記号の集合
 - $\Sigma \subset \Gamma$ はアルファベット: 入力記号
 - S は空でない有限集合であり,
その要素を状態とよぶ.
 - δ は $S \times \Gamma \rightarrow S \times \Gamma \times \{L, R\}$ なる関数
 - $s_0 \in S$ は特定の状態であり, 初期状態とよぶ.
 - $B \in \Gamma - \Sigma$ は特定の記号であり, 空白記号とよぶ
 - $F \subset S$ は特定の状態の集合であり, その要素を終了状態, もしくは受理状態とよぶ.

計算状況(時点表示)

- 無限長のテープの有限部分のみ空白でない記号が記入されている $a_1 \dots q_j a_i \dots a_k$
- 時点表示 $\sigma = (q_j, a_1 \dots a_i \dots a_k, i)$ または $a_1 \dots q_j a_i \dots a_k$

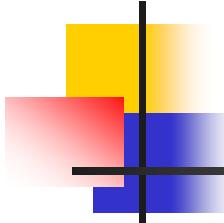


- $(q_j, a_1 \dots a_i \dots a_k, i) \Rightarrow (q_k, a_1 \dots a_l \dots a_k, l)$
遷移表を1回だけ用いて $\sigma = (q_j, a_1 \dots a_i \dots a_k, i)$ が
 $\tau = (q_h, a_1 \dots a_l \dots a_k, l)$ ($l = \pm i$) に遷移する



終端計算状況

- 計算状況 $(q_h, c_1 \dots c_i \dots c_k, i)$ が終端計算状況
 $\Leftrightarrow \delta(q_h, c_i)$ が定義されていない



句構造文法

- 句構造文法 $G = (N, \Sigma, P, S)$

N : 非終端記号(構文要素)の有限集合

Σ : 終端記号(単語・記号)の有限集合

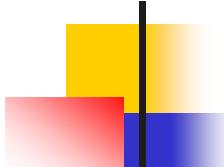
P : 生成規則の有限集合

$S \in N$: 開始記号

- 生成規則: $\alpha \rightarrow \beta$ という形の規則

$\alpha \in (N \cup \Sigma)^+, \beta \in (N \cup \Sigma)^*$

- α, β は非終端記号(構文要素)と終端記号(単語・記号)の有限列



句構造文法による導出

■ 句構造文法 $G = (N, \Sigma, P, S)$

列 $\gamma\alpha\delta$ ($\gamma, \alpha, \delta \in (\mathcal{N} \cup \Sigma)^*$) に対して生成規則 $\alpha \rightarrow \beta \in P$ が見つかれば, $\gamma\alpha\delta \Rightarrow \gamma\beta\delta$ とかく.

- 列 $\gamma\beta\delta$ は $\gamma\alpha\delta$ から $\alpha \rightarrow \beta$ によって **直接に導出される** という.
- 一つの列に α が 2 箇所以上みつかっても, それらを **同時に** β に 書換えることはできない.

$$S^* (S + S) \Rightarrow V^* (S + S)$$

$$S^* (S + S) \Rightarrow S^* (V + S)$$

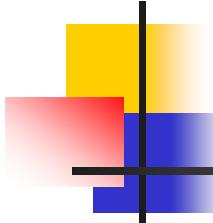
$$S^* (S + S) \not\Rightarrow V^* (V + V)$$

■ 列 α と β は, $\alpha = \beta$ であるか, または,

$$\alpha = \gamma_1 \Rightarrow \gamma_2 \Rightarrow \dots \Rightarrow \gamma_n = \beta$$

を満たす $\gamma_1, \gamma_2, \dots, \gamma_n$ が構成できるとき $\alpha \Rightarrow^* \beta$ とかく

- 列 β は α から G によって導出されるという.

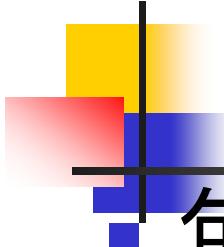


例(3)

$G_4 = (N = \{S, A, B, C, T\}, \Sigma = \{a, b\},$
 $P = \{S \rightarrow aBCT, S \rightarrow aBC,$
 $T \rightarrow ABCT, T \rightarrow ABC,$
 $BA \rightarrow AB, CA \rightarrow AC, CB \rightarrow BC,$
 $aA \rightarrow aa, aB \rightarrow ab, bB \rightarrow bb, bC \rightarrow bc,$
 $cC \rightarrow cc\}, S)$

$S \Rightarrow aBC \Rightarrow abC \Rightarrow abc$

$S \Rightarrow aBCT \Rightarrow aBCABC \Rightarrow aBACBC \Rightarrow aBABCC$
 $\Rightarrow aABBCC \Rightarrow aaBBCC \Rightarrow aabBCC \Rightarrow aabbCC$
 $\Rightarrow aabbcC \Rightarrow aabbcc$



句構造文法が生成する言語

句構造文法 $G = (N, \Sigma, P, S)$ に対して

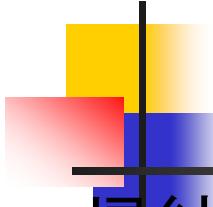
$$L(G) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}$$

を G が **生成する言語** という

- 言語 L は $L = L(G)$ なる句構造文法が存在すると
きに **帰納的に可算**(recursively enumerable) であ
るという

(注) 数学でいう“可算集合”とは、自然数全体の集
合 \mathbb{N} と 1 対 1 対応がある集合のことである。

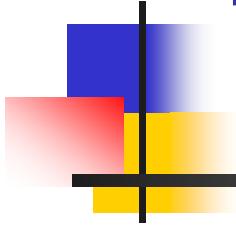
形式言語はすべて可算集合であるが、帰納的
に可算な言語とは限らない。



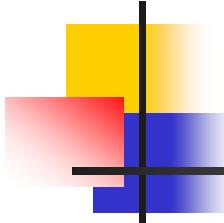
要素の枚挙(enumeration)

帰納的可算な言語の別定義

- 言語 L は自然数を入力とし, 記号列を出力する停止するTM M が存在して $\{w \in \Sigma^* \mid w=M(k), k \in \mathbb{N}\}$ となるとき, 帰納的可算であるという.
 - L の要素を M によって枚挙(enumerate)するという.
- 句構造文法 G の規則に附番しておくことで, 長さ m の導出で得られる $(N \cup \Sigma)^*$ の列に, 規則を順番に適用していくことで長さ $m+1$ の導出得られる $(N \cup \Sigma)^*$ の列を順番に生成することができる. この手順を $m=0$ から順に適用することで上の M が得られる(幅優先探索).



TMの停止性問題



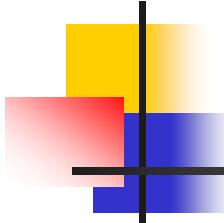
TMの停止性問題(1)

TMの停止性問題(Halting Problem) 任意に与えられた受理機械としてのTM M と語 $s \in \Sigma^*$ に対して $M(s)$ が停止するか？

注 この問題では“TMを生成する句構造文法” G を考え、入力となるTM M を $L(G)$ の語とみなしている。

- 実際、TM M は“各要素が $\langle q_k, c_h, D \rangle$ であるような表”である。以下では $[M]$ と表す。
- 表をテープ上に表現するには、(適当な個数の)新しいテープ記号を Γ に導入すればよい。
- さらに、 Γ の記号を (Σ の記号も含めて) 単純なアルファベット Σ' ($\{0, 1\}$ など) の記号列で表すことで、 $L(G)$ のアルファベットは Σ' であり、 $s \in (\Sigma')^*$ と仮定できる。

この仮定の下、以下でアルファベットは Σ としておく



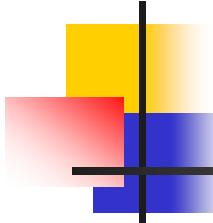
TMの停止性問題(2)

定理 TMの停止性問題は非可解である.

- TM M と 語 $s \in \Sigma^*$ の 対 $\langle [M], s \rangle$ を 入力 と し,
 $M(s)$ が 停止 する とき 受理 し,
 $M(s)$ が 停止 しない とき に は 停止 して 受理 し な い
よ う な TM M_H を 構成 す る こ と は で き な い

証明の方針: 背理法による

- $M_H(\langle [M], s \rangle)$ が 構成 でき た と して, M_H を 変形 し た
TM M'_H を つ く り, $M'_H([M'_H])$ を 考察 す る.



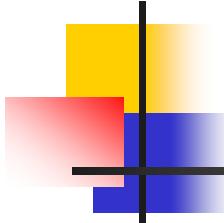
証明(1)

- TM M_H が構成できたとする。
 - M_H は任意の対 $\langle [M], s \rangle$ を入力とする。
 - $[M] \in \Sigma^*$ と仮定しているので, M_H は語の対 $\langle w, u \rangle$ を入力としている
- TM M_H から次のような M'_H を構成する
 - M'_H が $\langle w, w \rangle$ を受理するとき, M'_H は入力 w に対して停止しない(無限ループに入る)
 - M'_H が $\langle w, w \rangle$ を受理しないとき, M'_H は入力 w に対して停止して受理する
- M_H は必ず停止して受理しないことに注意



証明(2)

- TM M_H の入力 $[M_H']$ に対する動きは以下のようになる
- M_H' が入力 $[M_H']$ に対して停止しない(無限ループに入る)とすると、 M_H が $\langle [M_H'], [M_H'] \rangle$ を受理することになり、 M_H の定義に反する。
- M_H' が入力 $[M_H']$ に対して停止して受理するならば、 M_H は入力 $\langle [M_H'], [M_H'] \rangle$ に対して停止せずに受理しないことになり、 M_H の定義に反する。

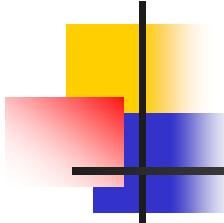


帰納的に可算でない言語

定理 (上の議論におけるアルファベットの仮定のもとで)

$L = \{[M] \in \Sigma^* \mid [M] \notin L(M)\}$ は帰納的可算ではない

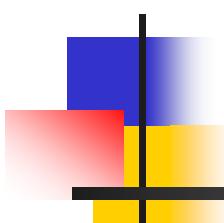
- アルファベットの議論を避けるためには、枚挙(enumeration)の考え方を用いる。
 - Σ^* 中のべての記号列を枚挙することができる
つまり, $\Sigma^* = \{w \mid w = W(k), k \in \mathbf{N}\}$ であるようなTM W を構成することができる
 - “TMを生成する句構造文法” G を考え, 入力となるTM M を $L(G)$ の語とみなすとき, $\{M \mid M = T(k), k \in \mathbf{N}\}$ がTM全体の集合となるようなTM T が構成できる.
 - このとき $L = \{W(k) \in \Sigma^* \mid W(k) \notin L(T(k)), k \in \mathbf{N}\}$ とする.



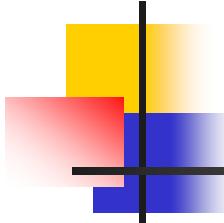
帰納的でない言語

定理 (上の議論におけるアルファベットの仮定のもとで)

言語 $L = \{\langle [M], w \rangle \mid w \in L(M)\}$ は帰納的可算であるが帰納的ではない.



アルゴリズムと決定可能性

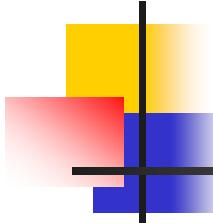


アルゴリズムとは

- 広義には:ある問題を解くための明確に記述された手続き(procedure)で停止するもの

例:多項式 $x^m + y^m = z^m$ を満たす正の整数 x, y, z と $m \geq 3$ の組を求めよ.

- 狹義には:“明確に記述された手続き”してTMを用いる
“ある問題を解くためのアルゴリズム”≡“停止するTM”

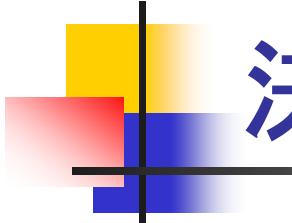


問題とインスタンス

- 計算の理論の対象は,
 - “入力のある問題に対するアルゴリズム”
 - ≡“どのような入力に対しても停止するTM”
 - 個々の入力を固定したときもとの問題の**インスタンス**という,

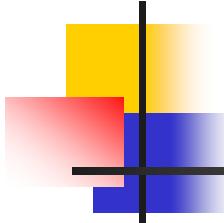
例:

- 問題“整数係数の任意の多変数の多項式 $p(x_1, \dots, x_n)=0$ が正の整数解を持つか?”(Hilbertの第10問題)の入力は個々の多項式 $p(x_1, \dots, x_n)$
- インスタンスは“多項式 $x^2y+3xy^2+5x=0$ は整数解を持つか? ”



決定可能性

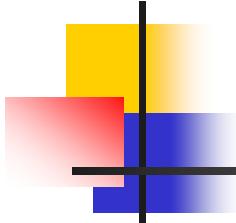
- 入力 w に対して答えが“Yes”か“No” か(0か1か)である問題 $P(w)$ が与えられたとき, 任意のインスタンス $P(s)$ に対してその答えを与えるアルゴリズム(**決定アルゴリズム**, decision algorithm)が存在するとき, $P(w)$ は**決定可能(可解, decidable)**であるという. 決定可能でないとき, **決定不能(非可解, undecidable)**という.



語の所属問題

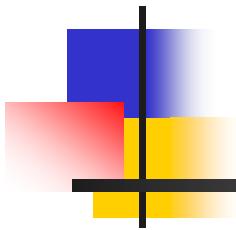
語の所属問題: あるクラスに属する任意の句構造文法 G と語 w が与えられたとき $w \in L(G)$ であるか?

- 帰納的言語とは、語の所属問題が可解である言語
- 語の所属問題が可解となる文法のクラス
 - 正則文法, 文脈自由文法, 文脈依存文法など
- 句構造文法全体からなるクラスは語の所属問題が可解とならない

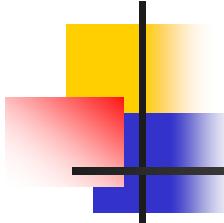


言語・文法・受理機械の階層

言語族	形式文法のクラス	受理機械の種類
帰納的可算	句構造文法	TM
帰納的		停止するTM
文脈依存	文脈依存文法	LBA
文脈自由	文脈自由文法	NPDA
決定性文脈 自由	LR(k)文法 ($k \geq 1$)	DPDA
正則	正則文法	FA



決定不能な問題



Postの対応問題(1)

- アルファベット Σ ($|\Sigma| \geq 2$) 上の記号列の列の対

$l_1 = [x_1, \dots, x_m], l_2 = [y_1, \dots, y_m]$ に対して, $x_{i_1} \dots x_{i_k} = y_{i_1} \dots y_{i_k}$ となるような添字の列 (i_1, \dots, i_k) を解とよぶ.

- l_1 と l_2 を構成する記号列の個数は同じ(m 個)
- $|x_i| = |y_i|$ である必要はない

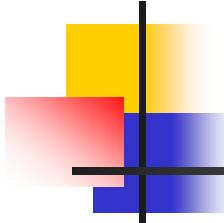
例1 $l_1 = [\text{aa}, \text{bb}, \text{abb}], l_2 = [\text{aab}, \text{ba}, \text{b}]$ に対して(1,2,1,3)は解

実際, aa bb aa abb = aab ba aab b

例2 $l_1 = [\text{a}, \text{bba}], l_2 = [\text{ab}, \text{bbba}]$ に対する解は存在しない

例3 $l_1 = [\text{a}, \text{abaaa}, \text{ab}], l_2 = [\text{aaa}, \text{ab}, \text{b}]$ に対して(2,1,1,3)は解

例4 $l_1 = [\text{ab}, \text{baa}, \text{aba}], l_2 = [\text{aba}, \text{aa}, \text{baa}]$ に対する解は存在しない

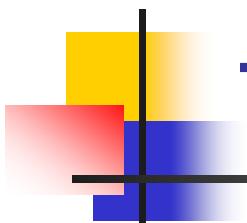


Postの対応問題(2)

Postの対応問題(Post's Correspondence Problem) 与えられた記号列の列の対 $l_1 = [x_1, \dots, x_m], l_2 = [y_1, \dots, y_m]$ に対して, 解(i_1, \dots, i_k)が存在するかどうかを判定する問題

定理 Postの対応問題は非可解である.

証明の方針 対 (x_i, y_i) がTMの遷移をシミュレートし,
TMが停止する \Leftrightarrow 解(i_1, \dots, i_k)が存在する
が成立するように l_1 と l_2 を構成する.



TMの遷移を表すPCPのインスタンス

x_i	y_i	
#	$\#q_0w\#$	遷移の開始
X	X	$X \in \Gamma$
#	#	
qX	Yp	$\delta(q, X) = (p, Y, R)$
ZqX	pZY	$\delta(q, X) = (p, Y, L), Z \in \Gamma$
$q\#$	$Yp\#$	$\delta(q, B) = (p, Y, R)$
$Zq\#$	$pZY\#$	$\delta(q, B) = (p, Y, L), Z \in \Gamma$
XqY	q	$q \in F, X, Y \in \Gamma$
Xq	q	$q \in F, X \in \Gamma$
qY	q	$q \in F, Y \in \Gamma$
$q##$	#	$q \in F$

例

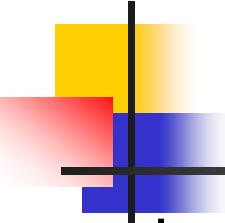
	F	B	a	b
q_0		$\langle q_1, b, L \rangle$	$\langle q_1, b, R \rangle$	$\langle q_1, a, L \rangle$
q_1		$\langle q_1, a, R \rangle$	$\langle q_2, a, L \rangle$	$\langle q_0, a, R \rangle$
q_2	O	λ	λ	λ

$w=ab$

x_i	y_i
#	$\#q_0ab\#$
a	a
b	b
#	#
$q_2\#$	##

q_0a	bq_1
aq_0b	q_1aa
bq_0b	q_1ba
$aq_0\#$	$q_1ab\#$
$bq_0\#$	$q_2bb\#$
aq_1a	q_2aa
bq_1a	q_2ba
q_1a	aq_0
$q_1\#$	$aq_1\#$

q_0a	q_2
aq_2a	q_2
aq_2b	q_2
bq_2a	q_2
bq_2b	q_2
aq_2	q_2
bq_2	q_2
q_2a	q_2
q_2a	q_2

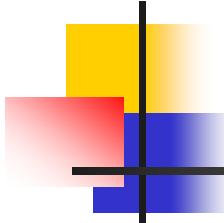


例(続き)

$$q_0 \mathbf{ab} \Rightarrow \mathbf{b} q_1 \mathbf{b} \Rightarrow \mathbf{b} \mathbf{a} q_0 \Rightarrow \mathbf{b} q_1 \mathbf{ab} \Rightarrow q_2 \mathbf{bab}$$
$$x_i : \#$$
$$y_i : \# q_0 \mathbf{ab} \#$$
$$x_i : \# q_0 \mathbf{a}$$
$$y_i : \# q_0 \mathbf{ab} \# \mathbf{b} q_1$$
$$x_i : \# q_0 \mathbf{ab} \# \mathbf{b}$$
$$y_i : \# q_0 \mathbf{ab} \# \mathbf{b} q_1 \mathbf{b} \# \mathbf{b}$$
$$x_i : \# q_0 \mathbf{ab} \# \mathbf{b} q_1 \mathbf{b}$$
$$y_i : \# q_0 \mathbf{ab} \# \mathbf{b} q_1 \mathbf{b} \# \mathbf{b} \mathbf{a} q_0$$

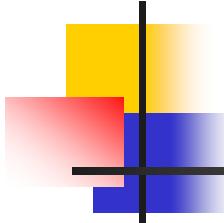
...

$$x_i : \# q_0 \mathbf{ab} \# \mathbf{b} q_1 \mathbf{b} \# \mathbf{b} \mathbf{a} q_0 \# \mathbf{b} q_1 \mathbf{ab} \# q_2 \mathbf{bab} \# q_2 \mathbf{ab} \# q_2 \mathbf{b} \# q_2 \# \#$$
$$y_i : \# q_0 \mathbf{ab} \# \mathbf{b} q_1 \mathbf{b} \# \mathbf{b} \mathbf{a} q_0 \# \mathbf{b} q_1 \mathbf{ab} \# q_2 \mathbf{bab} \# q_2 \mathbf{ab} \# q_2 \mathbf{b} \# q_2 \# \#$$



厳密には

- PCPに“解は必ず(1,...)という形でなければならぬ”という条件を加えても決定可能性は変わらない。
 - PCPの任意のインスタンス(L_1, L_2)に対して (L_1, L_2) が解を持つか否かと (L'_1, L'_2) が(1,...)という形の解を持つか否かが同値になるようなインスタンス(L'_1, L'_2)を構成することができる。

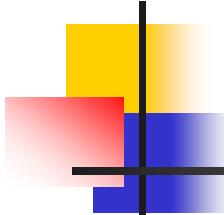


文脈自由言語の共通性問題

定理 任意に与えられた文脈自由文法 G_1, G_2 に対して,
 $L(G_1) \cap L(G_2)$ であるか否かは決定不能である

注 この問題では“文脈自由文法を生成する句構造文法” G を考え, 入力となる文脈自由文法 G_1, G_2 を $L(G)$ の語とみなしている.

証明の方針 $L(G_1) \cap L(G_2)$ であるか否かが決定可能であれば, Postの対応問題が可解となることを示す.



証明

- アルファベットを Σ ($|\Sigma| \geq 2$) とする.
- m 個の記号列の列 $l = [z_1, \dots, z_m]$ に対して, 文脈自由文法 $G(l) = (S, \Sigma', P, S)$ を次のように定める.

$$\Sigma' = \Sigma \cup \{d_1, \dots, d_m\} \quad (\Sigma \cap \{d_1, \dots, d_m\} = \emptyset)$$

$$P = \{S \rightarrow z_i S d_i, S \rightarrow z_i d_i \mid i=1, \dots, m\}$$

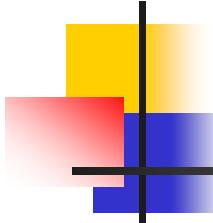
このとき

$$L(G(l)) = \{z_{i_1} \dots z_{i_k} d_{i_k} \dots d_{i_1} \mid k \geq 1\}$$

- PCP の任意のインスタンス ($l_1 = [x_1, \dots, x_m]$, $l_2 = [y_1, \dots, y_m]$) に対して

$$L(G(l_1)) \cap L(G(l_2)) \neq \emptyset$$

\Leftrightarrow PCP のインスタンス (l_1, l_2) は解をもつ



証明(1)

- アルファベットを Σ ($|\Sigma| \geq 2$) とする.
- PCPの任意のインスタンスを $([x_1, \dots, x_m], [y_1, \dots, y_m])$ に対して, $G_i = (\{S_i\}, \Sigma', P_i, S_i)$ ($i=1, 2$) を次のように定める.

$$\Sigma' = \Sigma \cup \{d_1, \dots, d_m\} \quad (\Sigma \cap \{d_1, \dots, d_m\} = \emptyset)$$

$$P_1 = \{S_1 \rightarrow x_i S_1 d_i, S_1 \rightarrow x_i d_i \mid i=1, \dots, m\}$$

$$P_2 = \{S_2 \rightarrow y_i S_2 d_i, S_2 \rightarrow y_i d_i \mid i=1, \dots, m\}$$

- すると $L(G_1) = \{x_{i_1} \dots x_{i_k} d_{i_k} \dots d_{i_1} \mid k \geq 1\}$

$$L(G_2) = \{y_{i_1} \dots y_{i_k} d_{i_k} \dots d_{i_1} \mid k \geq 1\}$$

なので $L(G_1) \cap L(G_2) \neq \emptyset$

\Rightarrow PCPのインスタンス $([x_1, \dots, x_m], [y_1, \dots, y_m])$ は
解をもつ



証明(2)

- 逆に, PCPのインスタンス($[x_1, \dots, x_m]$, $[y_1, \dots, y_m]$)が解 (i_1, \dots, i_k) をもつ \Rightarrow

$$w = x_{i_1} \dots x_{i_k} d_{i_k} \dots d_{i_1} = y_{i_1} \dots y_{i_k} d_{i_k} \dots d_{i_1} \in L(G_1) \cap L(G_2)$$

- PCPのインスタンス($[x_1, \dots, x_m]$, $[y_1, \dots, y_m]$)が解を持つ \Leftrightarrow 文脈自由言語の共通問題のインスタンス $L(G_1) \cap L(G_2) \neq \emptyset$ はYesである