文法と言語
ー字句解析とオートマトンlexー

和田俊和

資料保存場所

http://vrl.sys.wakayama-u.ac.jp/~twada/syspro/

前回の復習1

- 言語とは，あるルールに従う記号列の無限集合である.
- 文法を与えることで言語が定義できる.
- 文法にはタイプ0からタイプ3までのレベルがあり，
  - プログラム言語としてはタイプ2「文脈自由文法」
  - プログラム中の字句の表現にはタイプ3「正規文法」

  が用いられる.
- 文脈自由文法(Context Free Grammar：CFG)は，前後の記号に関係なく「非終端記号1つ」を「非終端記号と終端記号から成る記号列」に置き換えるという生成規則　A→t　のみを持つ文法
- 出発記号に対して生成規則の要素を何度か適用して終端記号列を得ることを「導出」と呼ぶ.
- 終端記号列と文法が与えられたとき，生成規則がどのように適用されたのか，つまり，導出の過程を求めることを「構文解析」と呼び，その結果，導出木が得られる.
- 導出木からそれを簡略化した「構文木（演算子木）」が求められ，それを利用した式の計算などが可能である.

前回の復習2

- 正規文法は，「非終端記号1つ」を「終端記号」もしくは「終端記号 非終端記号」に置き換えるという生成規則　A→a or B→bB　のみを持つ文法
- 正規文法で表現できるのは，「整数」「実数」「識別子（変数名）」「キーワード」など，比較的単純な記号の並びである.
- 正規文法によって規定される言語は，正規表現によって表現することができる.
- 正規表現から，それに唯一に対応付けられる「非決定性有限状態オートマトン（NFA）」が機械的に対応付けられる.
- その，機械的に求められたNFAは，計算機で実行可能な「決定性有限状態オートマトン（DFA）」に変換することができ，さらに状態数の最適化などが行われ，字句解析に用いられる.

オートマトンの利用例:字句解析

- 文字の連続的な並びに対して，字句の種類ごとに分類しつつ，単一の字句の切り出し（セグメンテーション）も行う.

例

- while (a14z < 100.4) a14z ...

予約語 左括弧 変数 比較演算子 実数 右括弧 変数名

- これは構文解析の前に行われる処理である.

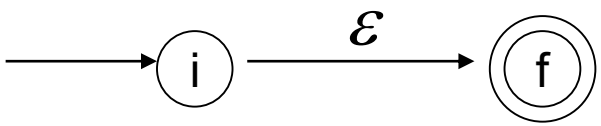
前回の演習課題

$$(1|2|...|9)(0|1|...|9)^*(\varepsilon|.(0|1|...|9)^*)$$

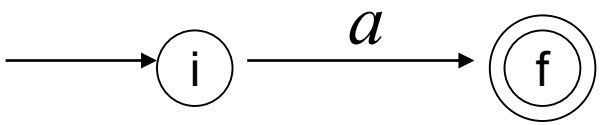
- 最初の数字は１～９までの数字であり,
- 引き続き０～９までの数字が0回以上繰り返される.
- これで終わりの場合もあるが,
- “. ”が来て０～９までの数字が0回以上繰り返される場合もある.

正規表現からオートマトンへの変換規則

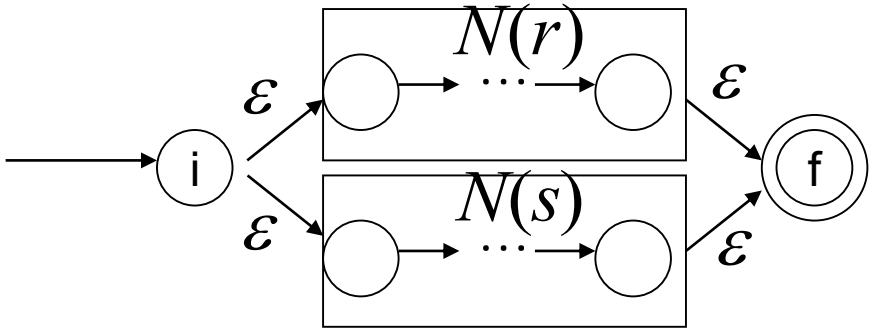
- ε



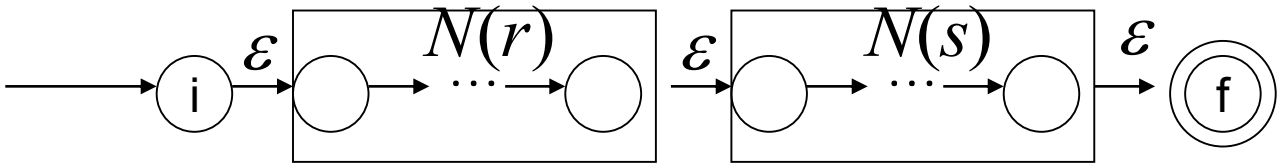
- a



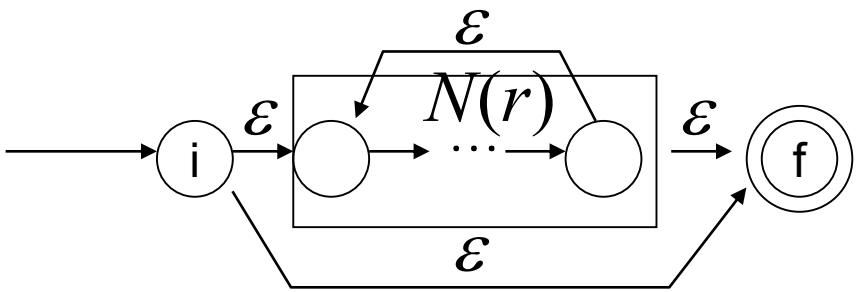
- $r \mid s$



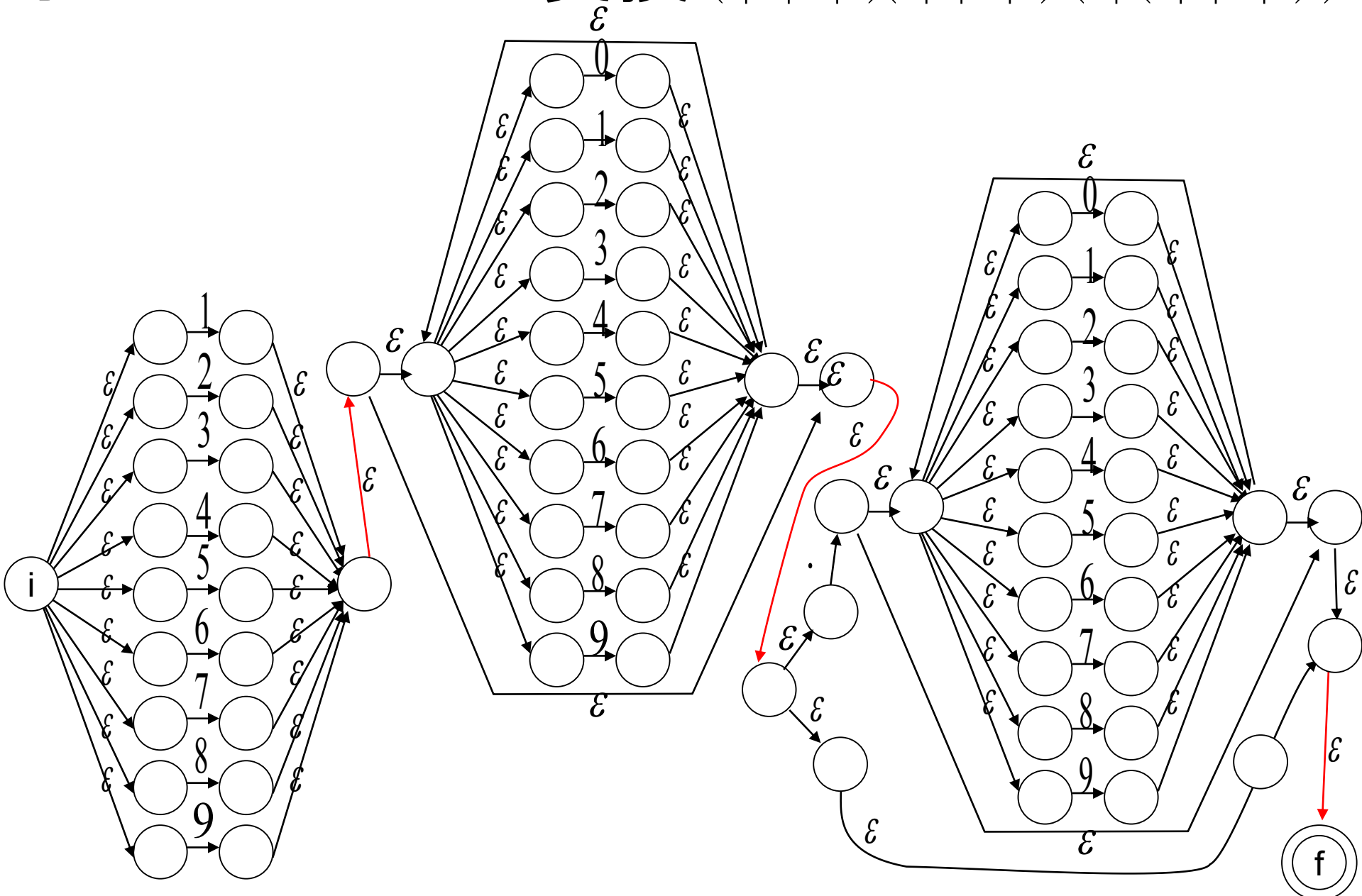
- rs



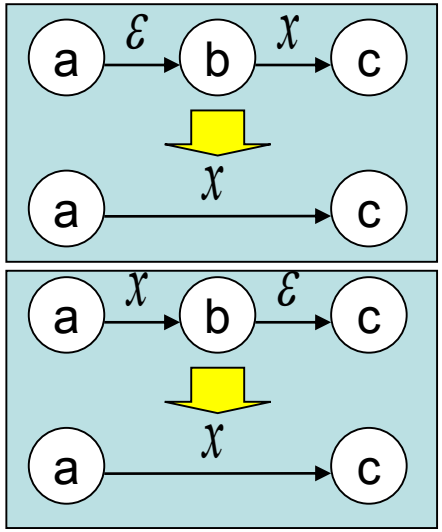
- r^*

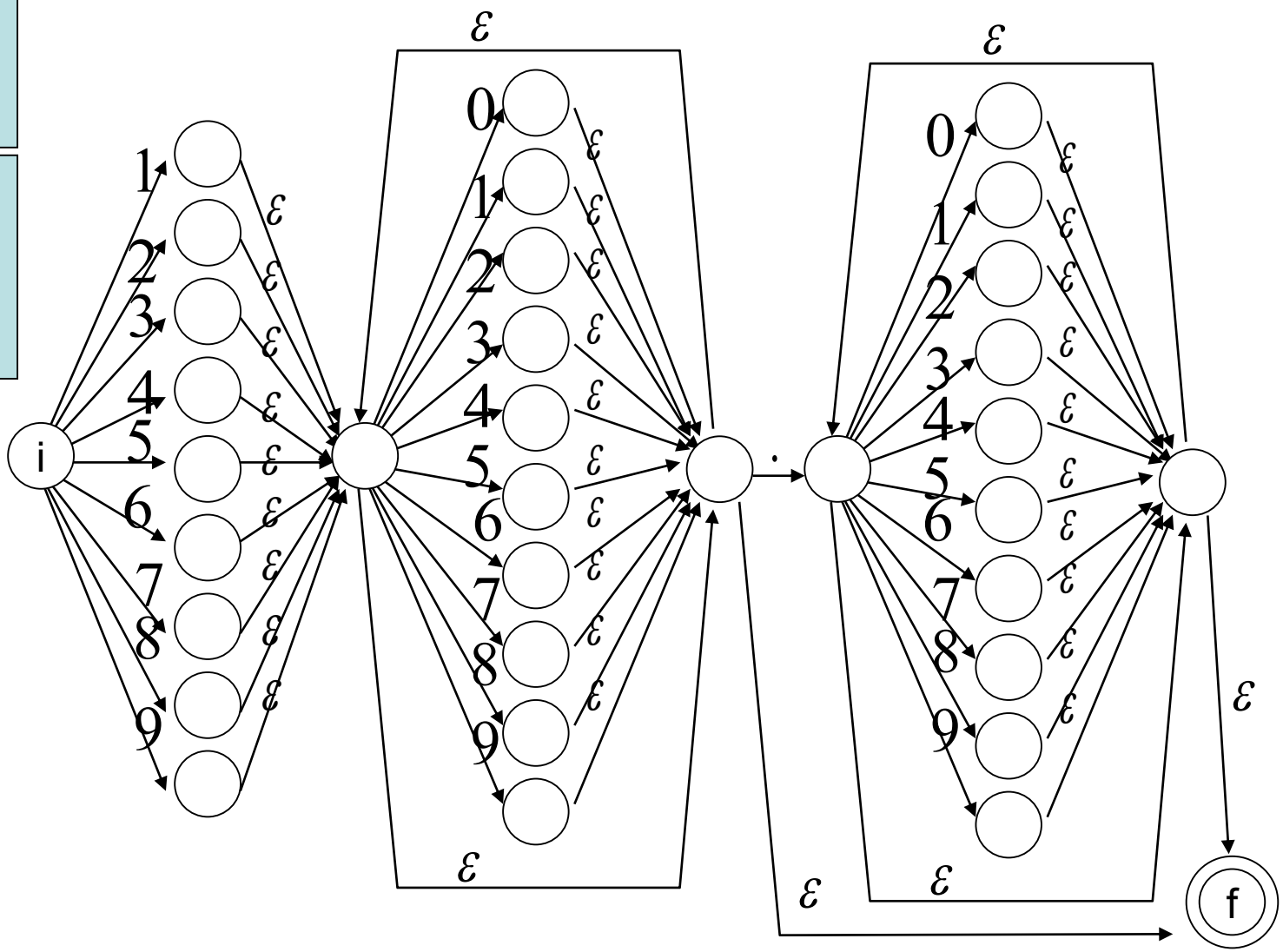


オートマトンへの変換 $(1|2|...|9)(0|1|...|9)^*(\varepsilon|.(0|1|...|9)^*)$



オートマトンへの変換(簡単化)





NFAからDFAへ

- NFA（非決定性有限状態オートマトン）がDFA（決定性有限状態オートマトン）と異なる点.
  - 「ε遷移」がある.
  - 同じ入力に対して複数の状態に遷移する「非決定性遷移」がある.

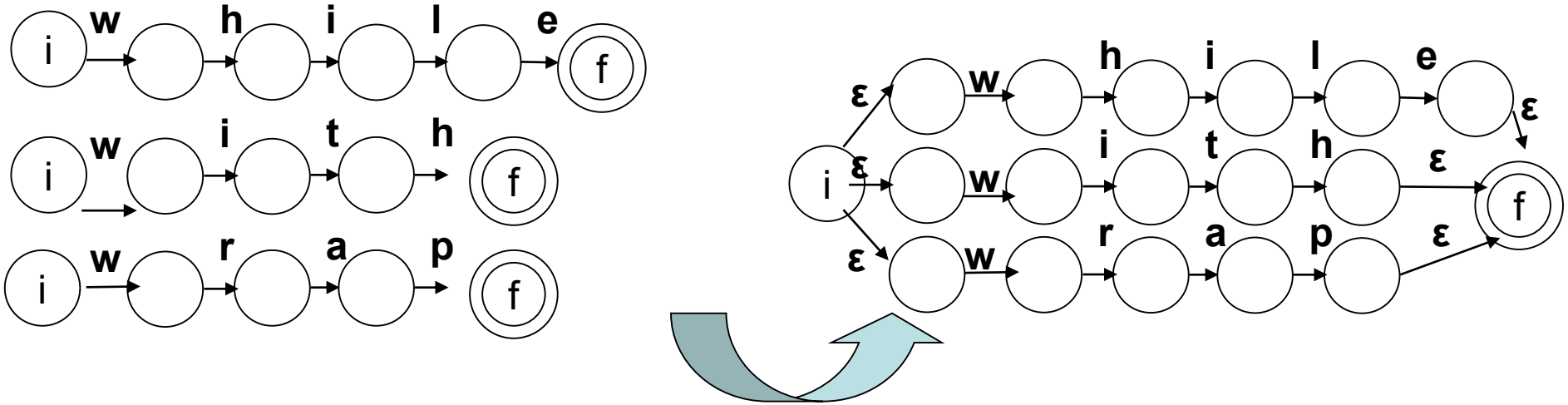
↓

これらを無くせば，DFAになる.

- 以下 “while”, “with”, “wrap” の3つの文字列を受理する3つのオートマトンを合成し，1つのNFAを作った後，これをDFAに変換する問題を例題として説明する.

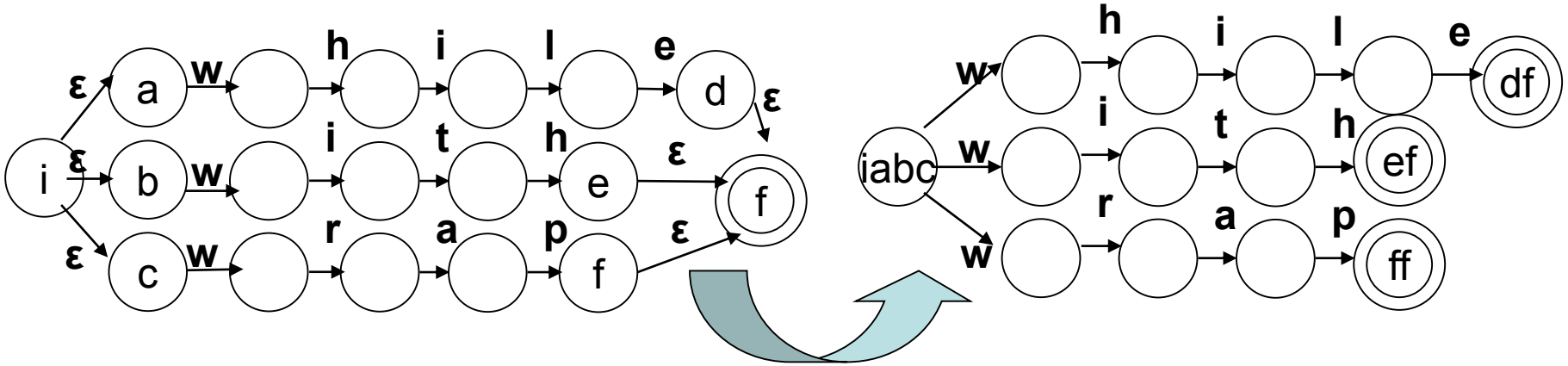
複数のオートマトンの結合

- while with wrapという記号列を受理するオートマトンから（while|with|wrap）という正規表現に対応するオートマトンを生成すると,「ε遷移」を含むオートマトンができる.



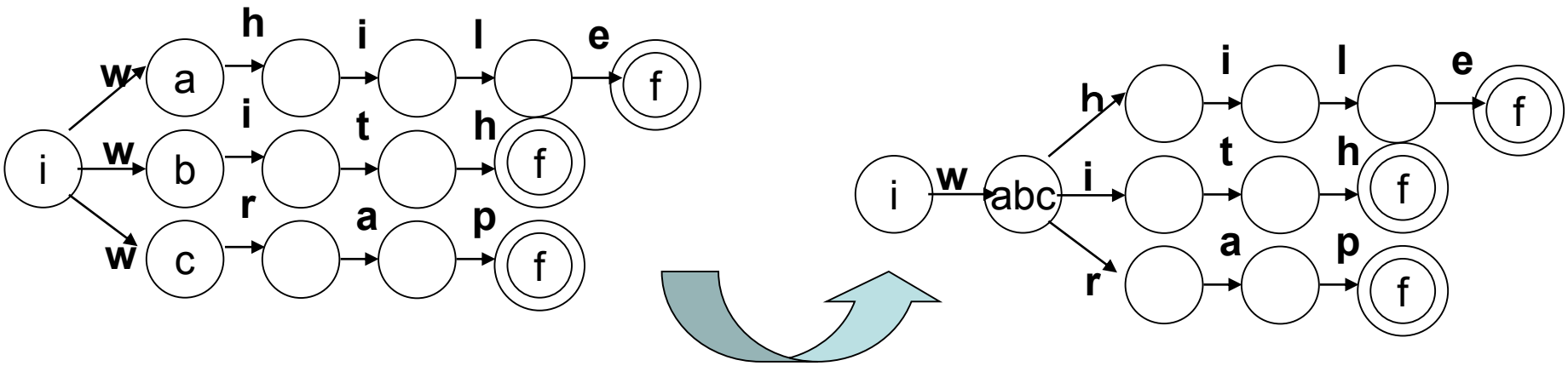
ε閉包（ε-closure）

- ある状態から無入力で遷移し得る状態の集合のこと.
- これら状態の集合を新たな状態として定義し直すことにより，ε遷移を含まないオートマトンができる.



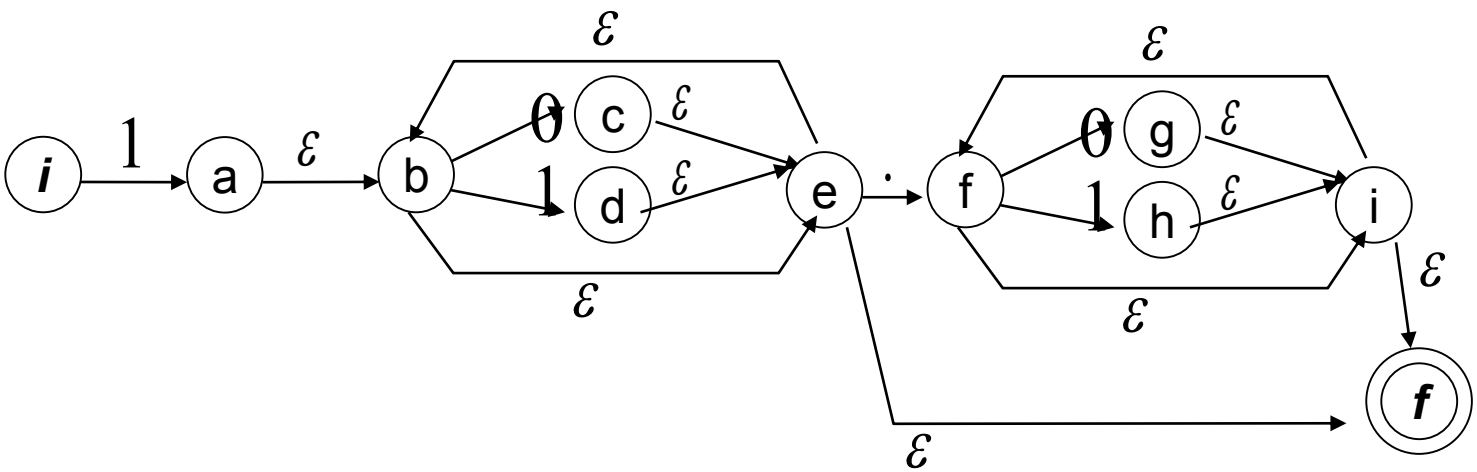
非決定的遷移の除去

- 同様に，ある状態に於いて，1つの入力記号が与えられたときに遷移し得る状態集合をまとめることにより，非決定的遷移も消すことができる.

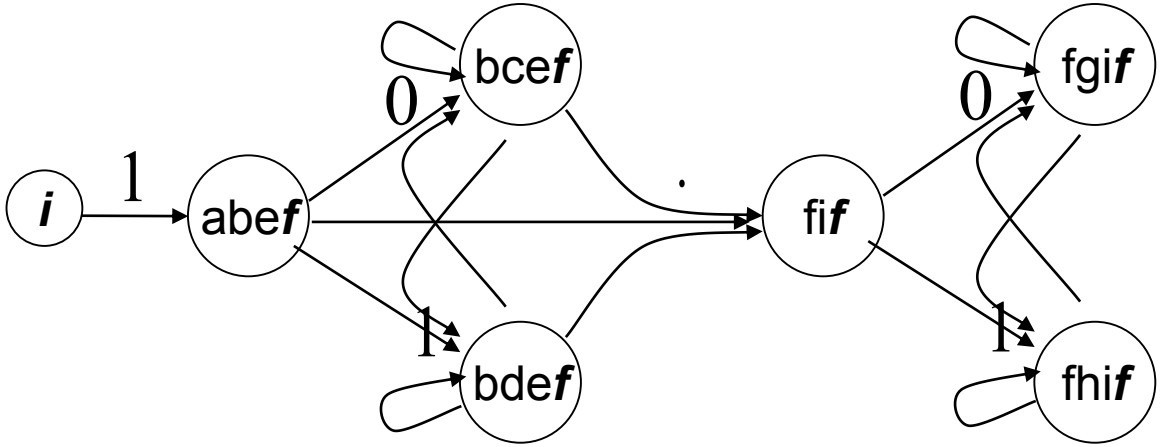


- これでDFAが得られたことになる.

このNFAを決定性に直す

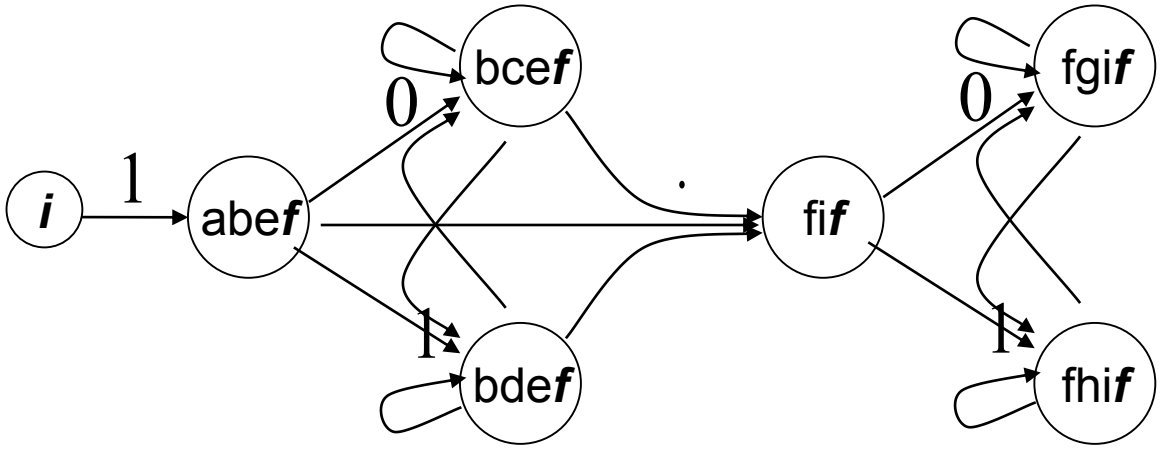


Non-deterministic Finite Automaton



Deterministic Finite Automaton

終了状態はどこにある？



Deterministic Finite Automaton

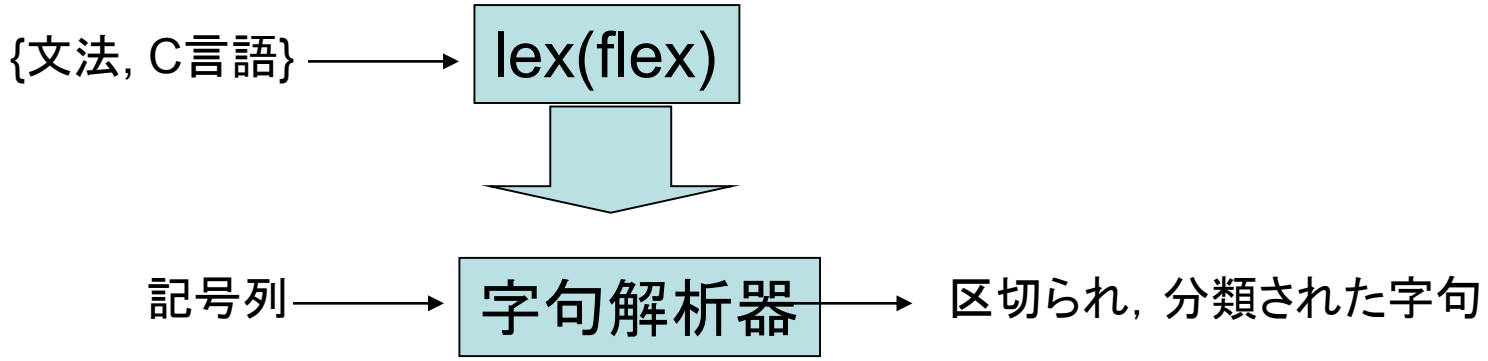
•元の終了状態 ***f*** を含む状態を，終了状態とする．→終了状態からの状態遷移は定義上おかしい．

•ε遷移で，終了状態に遷移させる．→DFAなのでε遷移はあってはいけない．

•各状態で受理しない記号を受け取った時点で終了状態に遷移する．→図として煩雑なので，ここでは明示しない．

実装は大変！
lex（flex）を使えばオートマトンも簡単に作れる.

- lex は，正規文法を与えて，それを解析するオートマトンを生成する，いわば「字句解析用オートマトン生成プログラム」である.
- 正規文法だけでなく，文字列を受理した際に行う処理をC言語で書くことができる.



flex プログラムと使い方

入力ファイルの構造

```
%{
任意のC プログラム。定数やC のマクロの宣言をここにかく。
%}
下の定義で使うlex のマクロの定義
%%
正規表現による入力パターンの定義
正規表現パターン アクション という形で書く
%%
任意のC プログラム
```

test.l

```
%{
#include <stdio.h>
%}
%%
a(b|c)* { printf("OK¥n"); } /*このパターンを入力したらOK を出力する*/
. {printf("NG¥n");} /*.は以上以外のパターンの場合のアクションを指定 */
%%
```

使い方

```
lex test.l
gcc lex.yy.c –ll
. /a.out
```

プログラム例

```
%{
#include <stdio.h>
%}
%%
a(b|c)* { printf("OK¥n"); } /*このパターンを入力したらOK を出力する*/
.     {printf("NG¥n");} /*.は以上以外のパターンの場合のアクションを指定 */
%%
```

実行例

```
./a.out
abcbabcbabcbbabbc
OK
OK
OK
OK
```

flex プログラムの例2

```
%{
/* You can put any C code here. */
#define END          0      /* end of input */
#define BADTOKEN     1      /* invalid token */
#define NUMBER       2      /* decimal number */
#define HEXNUMBER    3      /* hex number */
#define RETURN       4
#define PLUS         5
#define MINUS        6
%}
%%
[0-9]+                 return NUMBER;
0[Xx][0-9A-Fa-f]+      return HEXNUMBER;
“+”                    return PLUS;
“-”                    return MINUS;
“¥n”                   return RETURN;
[ ¥t]+                 ;      /* ignore */
.                      return BADTOKEN;
%%

#include <stdio.h>          /* for printf() */
#include <stdlib.h>         /* for atoi() */
```

```
int main()
{
    extern char* yytext;
    int token;

    while((token = yylex()) != END){
        printf("token no.%d: ", token);
        switch(token){
        case NUMBER :
            printf("%d¥n", atoi(yytext));
            break;
        case RETURN :
            printf("return¥n");
            break;
        default :
            puts(yytext);
            break;
        }
    }
}
```

正規表現と実行例

```
[0-9]+                  return NUMBER;
0[Xx][0-9A-Fa-f]+       return HEXNUMBER;
“+”                     return PLUS;
“-”                     return MINUS;
“¥n”                    return RETURN;
[ ¥t]+                  ;       /* ignore */
.                       return BADTOKEN;
```

```
-bash-3.1$ ./a.out
19+0x12+a          ←入力
token no.2: 19
token no.5: +
token no.3: 0x12
token no.5: +
token no.1: a
token no.4: return
```

本日の演習課題

$(what \mid where \mid when \mid why \mid who \mid how)$

という正規表現を受理するオートマトンを，

- 個々の文字列を受理するオートマトンをε遷移で束ねたNFA
- 決定性に変換したDFA

に分けて，それぞれ示しなさい．

A: a*b*
B: aa*b*
C: a*bb*
D: aa*bb*

右の正規表現に対応するオートマトンが受理する記号列（すなわち言語）L(A)～L(D)をVenn図で示しなさい．