

# ソフトウェア演習 1

## 第 4 回課題

J2200071 齊藤 隆斗

### 1. 課題 5.4, 5.9, 5.10 のプログラム

課題 5.4 関数 `get_dvertex()` 関数を与えよ.

課題 5.9 `check_dvertex` 関数中の、辺テーブルに基づいて DFA 頂点/辺を `dvlist` に登録する部分の処理を追加して、`check_dvertex` 関数を完成させよ.

課題 5.10 NFA から DFA を生成するプログラムを完成させ、オプションで指定することによって、生成された DFA をも表示できるようにメイン関数を変更せよ.

### 2. 実行結果

実行例 1: スライドの例について正常に動作するかどうかを確認

```
$ ./kadai4 -d4 'a|b*|c'
DFA
Number of DFA states: 4
Initial state: 0
(c) 0 => 3
(b) 0 => 2
(a) 0 => 1
(b) 2 => 2
Final states: 0 1 2 3
```

実行例 2: スライドの例について正常に動作するかどうかを確認

```
$ ./kadai4 -d4 'a.b*|(d.e)*'
DFA
Number of DFA states: 5
Initial state: 0
(d) 0 => 2
(a) 0 => 1
(b) 1 => 3
(e) 2 => 4
(b) 3 => 3
(d) 4 => 2
Final states: 0 1 3 4
```

実行例 3: その他の例についても正常に動作するかどうかを確認

```
$ ./kadai4 -d4 'a.\e.b'
DFA
Number of DFA states: 3
Initial state: 0
(a) 0 => 1
(b) 1 => 2
Final states: 2
```

実行例 4: その他の例についても正常に動作するかどうかを確認

```
$ ./kadai4 -d4 '(a|\0)*'
DFA
Number of DFA states: 2
Initial state: 0
(a) 0 => 1
(a) 1 => 1
Final states: 0 1
```

実行例 5: その他の例についても正常に動作するかどうかを確認

```
$ ./kadai4 -d4 '(a.b|c)*'
DFA
Number of DFA states: 4
Initial state: 0
(c) 0 => 2
(a) 0 => 1
(b) 1 => 3
(c) 2 => 2
(a) 2 => 1
(c) 3 => 2
(a) 3 => 1
Final states: 0 2 3
```

### 3. プログラムの流れの説明

今回の課題は NFA から DFA を生成するプログラムが主なテーマであるから、オプションとして d1, d2, d3 が入力された場合のプログラムの流れについては省略する. ここでは d4 がオプションとして入力された場合におけるプログラムの流れについて説明する. オプション d4 が選択された場合、プログラムは関数 `make_dfa()` を実行して停止する. したがって、関数 `make_dfa()` について見ていく. 関数 `make_dfa()` では、まず関数 `eval_expr()` によって、構文木を生成する. そして、この時に `curr_token` の値が `EOREG` であれば、正常に構文木を生成できたということであるから、次の処理に移る. `curr_token` の値が `EOREG` でなければ、正常に構文木を生成できなかったということであるから `parse_error()` を呼び出し、エラー処理を行う. 正常に構文木が生成された後は、生成した構文木を関数 `gen_nfa()` によって NFA へ変換する. そして、この変換した NFA を関数 `gen_dfa()` によって DFA へ変換する. この `gen_dfa()` については考察のセクションで扱う. DFA へ変換した後は DFA を表示し、確保していたメモリ領域を開放して、処理を停止する.

### 4. 考察

このセクションでは、関数 `gen_dfa()` についての説明を行う.

NFA を DFA へ変換するためには、改めて頂点を決定する必要がある. ここで、DFA の頂点は、NFA の頂点の集合であり、ある DFA の頂点があった場合、その頂点を構成するような NFA の頂点から同一の文字で遷移できるような NFA の頂点集合が DFA の頂点となる.

これを踏まえた上で、まずはじめに、`gen_dfa()` では、

```
dv = gen_dvertex(); /* dv == 0 */
enhance_nvset(initial_nv, dvlist[dv].nvset);
```

によって、DFA の初期状態を生成する. ここで、`gen_dvertex()` によって DFA の頂点を生成し、`enhance_nvset(initial_nv, dvlist[dv].nvset)` によって、NFA の初期状態と そこから  $\epsilon$  遷移で到達できる頂点の集合を、先程生成した DFA の頂点に与えていることに注意する.

次に、先ほど生成した初期状態の頂点から DFA を構成していく. 構成の手順は次のようである.

1. チェック済みでないような頂点があるかどうかを調べる.
  - ない場合、処理を停止する.
  - あった場合、その DFA の頂点から出るような辺と行き先の頂点を生成し、チェック済みでなかった頂点をチェックする.

上の処理を行うのが下のコード部分である.

```
while ( (dv = get_dvertex()) != NOT_FOUND ) {  
    check_dvertex(dv);  
    dvlist[dv].check = 1;  
}  
}
```

ここで、どのように DFA の頂点から出るような辺と行き先の頂点を生成するかについて考察する. ここで、辺テーブルというものを作成する. ラベルごとに、ある頂点から出発して遷移するような頂点集合によってこの表を埋めていく. 例として、以下の図のような NFA を考える.

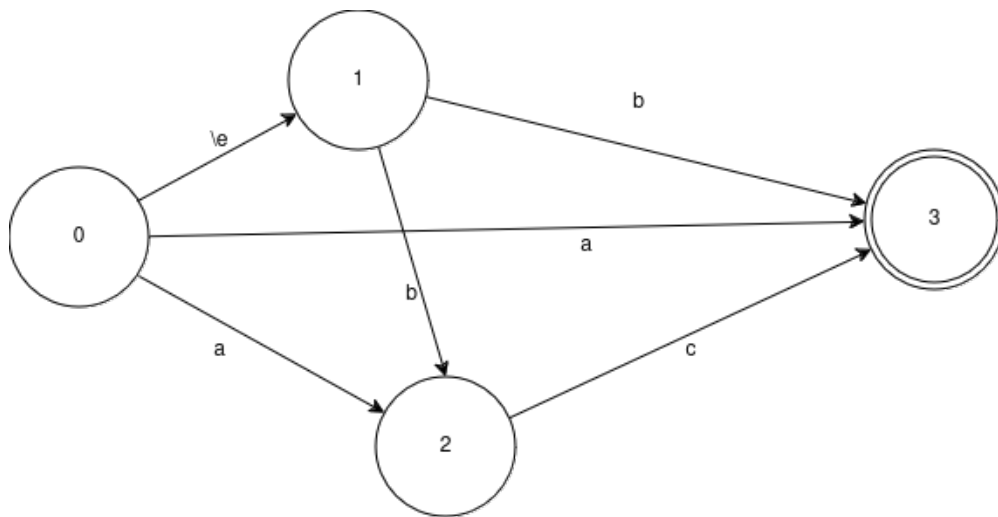


図 1: NFA

このとき、この図において頂点集合  $\{0, 1\}$  に関する辺テーブルは以下のように作成していく.

1. 0 から 3 へ遷移するラベル a の辺があり、まだ辺テーブルにラベル a の行はないので新たにラベル a で遷移先頂点集合が  $\{3\}$  の行を追加する.
2. さらに 0 から 2 へ遷移するラベル a の辺があり、すでに辺テーブルにラベル a の行が存在するので遷移先頂点集合に 2 を追加する.
3. 1 から 3 へ遷移するラベル b の辺があり、まだ辺テーブルにラベル b の行はないので新たにラベル b で遷移先頂点集合が  $\{3\}$  の行を追加する.
4. さらに 1 から 2 へ遷移するラベル b の辺があり、すでに辺テーブルにラベル b の行が存在するので遷移先頂点集合に 2 を追加する.

ただし、辺テーブルを作成する際、 $\varepsilon$  遷移がある場合には遷移先の頂点も頂点集合に追加されることに注意する. このようにして、以下の辺テーブルを作成することができる.

表 1: etable

出発頂点	辺	遷移先頂点集合
0	a	{2, 3}
1	b	{2, 3}

この表から、頂点集合  $\{0, 1\}$  から頂点集合  $\{2, 3\}$  へラベル a とラベル b によって遷移するような辺が 2 つあることがわかる. 辺テーブルを作成した後は以下の処理を行う.

1. 辺テーブルの行を上から 1 つずつ見ていき、該当行の頂点集合が構成中の DFA の頂点の頂点集合に一致するものが存在するかどうかを調べる.
  - 存在していない場合、新しく、該当行の頂点集合に対応する DFA の頂点を生成する.
  - 存在していた場合、何もせずに次の処理へ.
2. 関数 `add_dedge()` によって、遷移元頂点(上の例の場合、頂点集合が  $\{0, 1\}$  であるような頂点)から遷移先頂点(上の例の場合頂点集合が  $\{2, 3\}$  であるような頂点)へのラベル付の辺を設定する.
3. 辺テーブルのすべての行に対して上の 2 つの操作を繰り返し行う.

辺テーブルから辺と頂点の設定が終了したら、今まで遷移元の頂点としていた頂点にチェックを入れる. その後、まだチェック済みでない頂点に対して今までの処理を繰り返す.

上で述べた通りの処理によって NFA から DFA への変換が行われることを考察した.