ソフトウェア演習 1
第 3 回課題

J2200071 齊藤 隆斗

1. 課題 4.5, 課題 4.7-4.9 のプログラム

課題 4.5

頂点 nv1, nv2 がすであったとして、ラベル label をもつ、頂点 nv1 から頂点 nv2 への辺をグラフ追加する関数 add_nedge 作成せよ.

課題 4.7

残りの関数を記述せよ.

課題 4.8

ptree_to_nfa 関数を記述せよ.

課題 4.9

構文木から NFA 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、空遷移を含む NFA(ラベル付き有向グラフ+初期頂点+受理頂点) も表示できるように main 関数を変更せよ.

2. 実行結果

実行例 1: テキストの例について正常に動作するかどうかを確認

```
$ ./kadai3 -d3 'a|b*|c'
 a:   0 =>   1
\e:   1 =>  10
 b:   2 =>   3
\e:   2 =>   4
\e:   3 =>   2
\e:   4 =>   8
 c:   5 =>   6
\e:   6 =>   8
\e:   7 =>   2
\e:   7 =>   5
\e:   8 =>  10
\e:   9 =>   0
\e:   9 =>   7
Initial state:   9
Final state:  10
```

実行例 2: テキストの例について正常に動作するかどうかを確認

```
$ ./kadai3 -d3 'a.b*|c'
 a:   0 =>   1
\e:   1 =>   2
 b:   2 =>   3
\e:   2 =>   4
\e:   3 =>   2
\e:   4 =>   8
 c:   5 =>   6
```

```
\e:   6 =>   8
\e:   7 =>   0
\e:   7 =>   5
Initial state:   7
Final state:   8
```

実行例 3: EMPTY が含まれる場合に正常に動作するかどうかを確認

```
$ ./kadai3 -d3 '\0.a'
\e:   1 =>   2
 a:   2 =>   3
Initial state:   0
Final state:   3
```

実行例 4: EPSILON が含まれる場合に正常に動作するかどうかを確認

```
$ ./kadai3 -d3 '\e*'
\e:   0 =>   1
\e:   0 =>   2
\e:   1 =>   0
Initial state:   0
Final state:   2
```

実行例 5: 追加の例について正常に動作するかどうかを確認

```
$ ./kadai3 -d3 '(a|b)*.c'
 a:   0 =>   1
\e:   1 =>   5
 b:   2 =>   3
\e:   3 =>   5
\e:   4 =>   0
\e:   4 =>   2
\e:   4 =>   6
\e:   5 =>   4
\e:   6 =>   7
 c:   7 =>   8
Initial state:   4
Final state:   8
```

3. プログラムの流れの説明

今回のプログラムでは構文木から NFA に変換するようなプログラムを作成したので、d1, d2オプションが指定された場合のプログラムの流れは省略する. ここでは、d3 オプションが指定された場合、すなわち構文木を NFA へ変換し、 表示する関数 make_nfa()についてプログラムの流れについて説明する.

まず、第一回の課題で作成した関数 get_token()で、１つ目のトークンを取得する. その後、第 2 回課題で作成した関数 eval_expr()によって、そのトークンから構文木を生成する. この構文木が生成し終わった時点で curr_token の値が EOREG でない場合は解析エラーであるから、parse_error()関数によってエラー処理を行う. 正常に解析ができた場合は、関数 gen_nfa()によって NFA を構成する. この関数 gen_nfa()の詳細な流れは考察で触れる.その後、構成した NFA を表示する. そして、構文解析木と NFA で確保していたメモリを開放し、処理を停止する.

4. 考察