課題 2(b)：課題

NFA から DFA を構成するサブセット構成を実装する．

DFA の実装の説明

DFA は，以下のように状態の型 Q, アルファベットの型 A に関して，多相的なクラスとして実装している．

- アルファベット A 上の文字列は，A 上のリストで表現している．
- 遷移関数は, Scala の Map を用いて, 状態とアルファベットの組から状態へのマップとして定義している．
  - 数学的な定義とは異なり，Map を用いて定義しているので，ある状態 q と記号 a に対して，$\delta(q, a)$ が未定義なことがある．
  - Scala の Map や Set の使い方は，ネットで検索してください．この資料の最後に簡単な使用例を書いておきました．
- メソッド trans(q, w) は，状態 q で入力として文字列 w を受け取った時に，どの状態に遷移するかを返す関数である．
- メソッド accept(w) は，文字列 w を受理するか判定する関数である．ソースコードを見ると，例外に関する処理が必要となっている．

```
class DFA[Q, A](
  val states: Set[Q],
  val alpha: Set[A],
  val transition: Map[(Q,A),Q],
  val q0: Q,
  val finalStates: Set[Q]) {

  def trans(q: Q, w: List[A]): Q = ...

  def accept(w: List[A]) = ...
}
```

NFA の実装の説明

NFA も，以下のように状態の型 Q, アルファベットの型 A に関して，多相的なクラスとして実装している．

- 遷移関数の定義では, Option 型を用いている．
  - ϵ を None で, $a \in \Sigma$ を Some(a) で表現している．

```
class NFA[Q,A](
  val states: Set[Q],
  val alpha: Set[A],
  t: Map[(Q,Option[A]),Set[Q]],          // ε を None で表現
  val q0: Q,
  val finalStates: Set[Q]) {

  ...

  def eclosure(aQs: Set[Q]) = {
    var qs = Set[Q]()                   // Q 型の空集合を作成
    var newQs = aQs
    while (newQs != qs) {
      qs = newQs
```

```
      for (q <- qs) newQs = newQs ++ transition((q, None))
    }
    qs
  }
}
```

課題

NFA の実装の中で，サブセット構成で NFA から DFA を構成するプログラムの雛形を与えている．このプログラムを完成させてください．雛形を用いなくても良い．

```
  def toDFA(): DFA[Set[Q],A] = {
    val q0DFA =eclosure(Set(q0))
    var statesDFA = Set(q0DFA)
    var u = List(q0DFA)            // リストをスタックとして使用
    var transitionDFA = Map[(Set[Q], A), Set[Q]]()

//    while (!u.isEmpty) {
//       ...
//     }
    val finalStatesDFA = statesDFA.filter(qs => !(qs & finalStates).isEmpty)

    new DFA(statesDFA, alpha, transitionDFA, q0DFA, finalStatesDFA)
  }
```

sbt でテストを実行すると，DFA と NFA の実装に関するテストが実行される．サブセット構成を実装しないで，テストを実行すると以下のように最後のテストでエラーになる．

```
> test
[info] Compiling 1 Scala source to /Users/minamide/lectures/compiler/kadai/automata-sbt/target/scal
[info] Compiling 1 Scala source to /Users/minamide/lectures/compiler/kadai/automata-sbt/target/scal
[info] DFATest:
[info] DFA
[info] - should 遷移
[info] NFATest:
[info] NFA
[info] - should ε-closure
[info] NFA
[info] - should tansition
[info] NFA
[info] - should accept
[info] NFA
[info] - should サブセット構成 *** FAILED ***
[info]   Set(Set(0, 1, 2)) did not equal Set(Set(0, 1, 2), Set(1, 2), Set(2), Set()) (Automata.scal
```

サブセット構成を実装できたら，テストを追加して実行結果を確認してください．

オプション課題

RegExp.scala に，以下のように正規表現を表すための型を定義している．RegExp を NFA に変換するプログラムを書け．

```
trait RegExp
case class CharExp(c: Char) extends RegExp
case object EmptyExp extends RegExp      // 空集合
case object EpsExp extends RegExp        // ε
case class ConcatExp(r1: RegExp, r2: RegExp) extends RegExp // 連接
```

```
case class AltExp(r1: RegExp, r2: RegExp) extends RegExp    // 選択
case class StarExp(r: RegExp) extends RegExp          // 繰り返し
```

提出するファイル

以下のファイルの中で，変更したものを kadai2b.zip というファイル名で，まとめて提出すること．

```
src/Automata.scala
src/RegExp.scala
test/Automata.scala
```

追加したファイルがあれば追加したファイルも含めてください．

Scala の機能の紹介

クラスを定義する時に，コンストラクタの引数に `val` を付加することで，その引数がインスタンス変数として使えるようになる．下に例を示す．

プログラム例

```
class A (x: Int) {
  val y = x * x
}

class B (val x: Int) {
  val y = x * x
}
```

実行例

```
scala> val a = new A(10)
scala> a.x
a.x
<console>:13: error: value x is not a member of A
scala> val b = new B(10)
scala> b.x
res5: Int = 10
```

- クラス `A` のオブジェクトを作成し，インスタンス変数 `x` を参照しようとするとエラーになる．
- クラス `B` の場合, `val` が付加されているのでインスタンス変数として扱われている．

Set と Map

`Set` と `Map` の使い方を例で簡単に説明する．

```
scala> Set(1,2) == Set(2,1)
res9: Boolean = true

scala> val x = Set(1,2,3,4)
x: ... = Set(1, 2, 3, 4)
scala> val y = Set(2,4,6)
y: ... = Set(2, 4, 6)

scala> x.contains(1)        // 要素として含むか判定
res15: Boolean = true
```

```
scala> x.contains(5)
res16: Boolean = false

scala> x ++ y                // 和集合
res1: ... = Set(1, 6, 2, 3, 4)

scala> x & y                 // 共通部分
res0: ... = Set(2, 4)

scala> x.map(z => z * z)     // map: すべての要素に関数を適用
res4: ... = Set(1, 4, 9, 16)
```

Set[a] 型の集合に a => Set[b] 型の関数を適用して，Set[b] 型の集合を作る．

```
scala> x.flatMap(z => Set(z, z * z))
res3: ... = Set(1, 9, 2, 3, 16, 4)
```

次に Map の使用例を示す．

```
scala> val m = Map("a"->1,"b"->2)
m: ... = Map(a -> 1, b -> 2)

scala> m.apply("a")   // マップを適用
res12: Int = 1
scala> m("a")         // 関数適用の形でも書ける
res13: Int = 1

scala> m+("c"->3)     // 対応を追加
res10: ... = Map(a -> 1, b -> 2, c -> 3)

scala> val m0=Map[String,Int]()  // 空のマップ
m0: ... Map[String,Int] = Map()
scala> m0+("a"->1)
res11: ... = Map(a -> 1)
```

次に DFA や NFA での使い方に近い例を示す．

```
scala> val m1 = Map[(Int,String), Int]()
m1: ... Map[(Int, String),Int] = Map()

scala> m1 + ((1,"a") -> 2)
res18: .... Map[(Int, String),Int] = Map((1,a) -> 2)
```