RareSequence operator

The RareSequence operator finds seldom sequences in data streams. It's build for discrete values, e.g. states. It is important to have recurring tuples and therefore remove attributes from the tuples which make "equal" tuples unique. E.g., if you have a tuple with a counting number and a recurring state [(1, "state x"), (2, "state y"), (3, "state x"), (4, "state x"), ...], this operator won't work. In this case, you would need to use a projection to remove the counter to get tuples like [("state x"), ("state y"), ("state x"), ("state x")] with each "()" as a single tuple.

The operator builds a tree from the incoming tuples and counts for each node, how often it was used. With this information, it can calculate the probability for each node in comparison to its siblings and therefore for each path in the tree. If the probability is below the value given by the user, the current tuple is marked as anomaly and will be reported.

Parameters

- **treeDepth** The maximum depth of the tree. Default is 2.
- **minRelativeFrequencyPath** The minimal relative frequency of that path. The default is 0.3 (which means 30%).
- **minRelativeFrequencyNode** The minimal relative frequency of all single nodes of the path. The default is 0.3 (which means 30%)
- **firstTupleIsRoot** If true, tuples which are equal to the first tuple are always the root. It could be good to set the maxdepth higher than the expected number of states between two occurences of the root state.
- **uniqueBackupId** A unique ID for this operator to save and read backup data.

Example

```
stateAnalysis = RARESEQUENCE({
                    treedepth = 100,
                    minrelativefrequencyPath = 0.1,
                                    minrelativefrequencyNode = 0.3,
                                    firsttupleisroot = 'true'
                  },
                  state
                )
```

Using the backup functionality

The operator can save the learned tree into a database via the second output port. It continuously put out a serialized version of the tree which can be used to save it in a database, e.g. a MongoDB. When the analysis is started again, the information from the database can be used. This is useful, if a lot of knowledge is learned and if it would take a while to gain this knowledge again. On the startup, the information from the database is read a single time (if an information is available) and written into the operator. The PQL query in the codeblock shows how to use the backup functionality.

```
#PARSER PQL
#DEFINE BACKUPSCHEMA [['tree', 'String'],['backupId', 'String']]
#RUNQUERY
/// Backup-Daten aus der Datenbank auslesen
backupMongo = MONGODBSOURCE({
                  database = 'odysseus',
                  port = 27017,
                  host = 'localhost',
                  collectionname = 'condition'
                }
              )
/// Backup-Daten in Tupel konvertieren
backupTuple = KEYVALUETOTUPLE({
            schema=${BACKUPSCHEMA},
            TYPE = 'Backup',
            KEEPINPUT = 'false'
          },
          backupMongo
        )
stateAnalysis = RARESEQUENCE({
                    treedepth = 100,
                    minrelativefrequencyPath = 0.1,
                    firsttupleisroot = 'true',
                    UNIQUEBACKUPID = 'rareSequence_1'
                  },
                  state,
                  backupTuple
                )

analysis = SELECT({PREDICATE = 'anomalyScore > 0'}, stateAnalysis)

/// Backup-Daten von Port 1 in ein Key-Value Objekt konvertieren
keyValueOp = TUPLETOKEYVALUE({
                  type='KEYVALUEOBJECT'
                },
                1:stateAnalysis
              )

/// Backup-Daten in der Datenbank speichern
mongoSink = MONGODBSINK({
                database = 'odysseus',
                port = 27017,
                host = 'localhost',
                collectionname = 'condition',
                batchsize = 1,
                deletebeforeinsert = 'true',
                deleteequalattribute = 'backupId'
              },
              keyValueOp
            )
```