Spark-Homework: Transformation pipeline

Input Tuples --> Cells

- Splitting each record with (value, (Singleton type))

```
datasets.flatMap, columns.map, cells.reduce,
groupedCellsTuples.toDF
--------------------------------------------------
```

Cells --> Cache-based Preaggregation (optional when repeatedly occurring cells with values)

- Should reduce network load
- Groups by the value, and the singleton type

```
.groupBy(value).agg(collect_set(singletonType)
```

```
--------------------------------------------------
```

Cache-based Preaggregation --> Global partitioning

- Reordering the cells among the workers of the cluster through hashing
- *We need an appropriate function to map each different value to unique cell*

$$p(v) \overset{def}{=} hash(v) \bmod n.$$

- Therefore: cells with the same value are on the same worker

```
Done by spark
```

Spark-Homework: Transformation pipeline

Global partitioning --> Attribute Sets

- Grouping all cells by their values
- Aggregating attribute sets using union operator

```
.select("aggregatedAttributeSet"), .distinct()
```

```
----------------------------------------------------------
```

Attribute Sets --> Inclusion Lists

- Set with n attributes = n inclusion lists (all possible combinations)

```
.select(explode(aggregatedAttributeSet),
.map(row => (row._1, row._2.toList.filter(_ != row._1)))
```

```
------------------------------------------------------------
```

Inclusion List --> Partition

- Group by the first attribute

```
Done by spark
```

```
------------------------------------------------------------
```

Partition --> Aggregate

- Intersection with preaggregation
- Ends with attributes with empty sets; no (n,0)

```
.groupBy(firstAttribute).agg(collect_set(inclusionArray))
.map(row => (row._1,row._2.reduce(_.intersect(_))))
.filter(row=> row._2.nonEmpty)
.sort(firstAttribute)
```

```
----------------------------------------------------------
```

Aggregate --> INDs

- Disassembling into INDs

```
.collect()
.foreach(row => println(row._1 + " < " + row._2.reduce(_ +
", " + _)) )
```