Specification of Genomic Data Model (GDM) and Genometric Query Language (GMQL)

Genomic Computing Group

Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano

November 27, 2015

Contents

1 Introduction

In this report, we provide the specification of the **Genomic Data Model** (GDM) and of the **Genometric Query Language** (GMQL). GDM describes biological samples by means of two fundamental abstractions, one for genomic regions and one for meta-information.

- Genomic regions are characterized by their coordinates relative to a reference alignment; in such regions, biological samples exhibit relevant information (e.g., specific DNA sequences, in the form of DNA encodings; or gene mutations, in the form of variations from gene sequences; or peaks of expression, further characterized by geometric or statistical properties).
- The meta-information of each sample is defined by free lists of attribute-value pairs, which allow the identification of generic properties of the experiment (e.g., experimental condition, cell line, biological sample, or the kind of disease and patient phenotype when data have clinical nature).

GMQL raises the level of data abstraction with regard to current bioinformatics query languages, as it allows query formulation by means of powerful and at the same time rather simple operators; thus, the language supports knowledge discovery along a variety of dimensions. The name *genometric* indicates that an important aspect of the query language is to deal with genomic distances, which are measured in terms of bases (nucleotides) between regions of the genome; as we assume that genomic data are aligned to known references, such distances are computed as simple arithmetic operations between region coordinates.

In the long range, the main difficulty in dealing with NGS data will be scalability to thousands or even millions of experiments; therefore, the design of GMQL is inspired by *Pig Latin* [3], a high-level, declarative language that can be executed over an *Hadoop* cloud computing architecture [4]. We focus on providing a model and query language for data interoperability at large scale, across experiments; interoperability is guaranted by the use of a type system to express region data, so that compatible types support well-defined operations between them, and on arbitrary attribute-value pairs for metadata.

We do not want either to replace the current formats for genomic data storage or to replace the data analysis tools that are used by scientists; rather than interfering with the current experimental pipelines that are available at each research center, our work is placed downstream of such pipelines and uses their results and formats as inputs, by providing outputs in current standard bioinformatics formats.

GDM and GMQL are immediately available to scientists; for this reason, we rewrite query results in the standard GTF format [5] so that they can be subject of further analysis using conventional tools, or communicated using classic methods (e.g., Distributed Annotation System (DAS) protocol [6]), or observed by using classic viewing methods (e.g., University of California at Santa Cruz (UCSC) Genome Browser [2] or Integrated Genome Browser (IGB) [7]). Note that query results have normally a small size even with very large datasets, and therefore such rewritings require negligible execution time and efforts.

2 Genomic Data Model

The genomic data model represents genomic data samples; each sample, in turn, consists of two parts, the *region data*, which describe genomic regions of the sample, aligned to a specific reference, and the *metadata*, which describe other properties of each sample, not specifically related to genomic regions.

The key aspect of the model is the notion of genomic region. A **genomic region** r_i is a well-defined portion of the genome, further qualified by a quadruple of values, called **region coordinates**: $r_i =< chr, left, right, strand >$ where chr is the chromosome id, $left$ and $right$ are the two ends of the region along the DNA coordinates; $strand$ represents the direction of DNA reading, encoded as either $+$ or $-$, and can be missing (encoded as $*$)[1]. Thus, a region r_i corresponds to all the DNA nucleotides whose position is between its left and right ends; however, in general, we do not include DNA sequences within region data, but rather we store high-level properties of the region [2]. Thus, each dataset has specific attributes describing its region properties, and each attribute has an elementary type.

Metadata describe the biological and clinical properties associated to each sample; due to the great heterogeneity of information that can be associated with each sample and/or dataset, they are represented as arbitrary attribute-value pairs. We expect metadata to include at least the experiment type, the sequencing and analysis method used for data production, the cell line, tissue, experimental condition (e.g., antibody target) and organism sequenced; in case of clinical study, individual's descriptions including phenotypes.

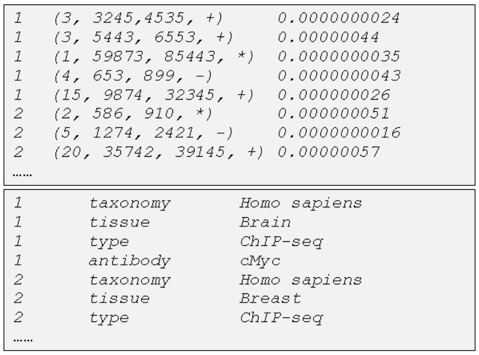
2.1 Formal definition

Formally, a **sample** s is a triple $< id, \{< r_i, v_i >\}, \{m_j\} >$ where:

- id is the sample identifier, of type `long`.
- Each region is a pair of **coordinates** r_i and `values` v_i; coordinates are four fixed attributes `chr, left, right, strand`, which are typed `string, int, int, string`; values are typed attributes; we assume attribute names of a sample to be different, and types to be any of `string, int, long, real, Boolean`. The `region schema` of s is the set of attribute names used for coordinates and values; the `region type` of s is the record of the corresponding elementary types

[1] According to the UCSC notation, we use *1-based interbase coordinates*, i.e., the considered genomic sequence is $[left, right)$. Left and right ends can be identical (e.g., when the region represents deletion mutations).

[2] E.g., with ChIP-seq experiments, whose regions describe peaks of protein binding, the region type is a record describing peak's properties, such as *p-value* and *q-value*; with transcription factors, whose regions describe binding sites, the region type includes a string of characters encoding the DNA motif. With DNA-seq mutation data, whose regions describe mutations, the region type includes the *mutation sequence* and possibly its *class*.



1	(3, 3245,4535, +)	0.0000000024
1	(3, 5443, 6553, +)	0.00000044
1	(1, 59873, 85443, *)	0.0000000035
1	(4, 653, 899, -)	0.0000000043
1	(15, 9874, 32345, +)	0.000000026
2	(2, 586, 910, *)	0.000000051
2	(5, 1274, 2421, -)	0.0000000016
2	(20, 35742, 39145, +)	0.00000057
……		

1	taxonomy	Homo sapiens
1	tissue	Brain
1	type	ChIP-seq
1	antibody	cMyc
2	taxonomy	Homo sapiens
2	tissue	Breast
2	type	ChIP-seq
……		

Figure 1: Regions and metadata of a dataset consisting of two samples; for ease of reading, region coordinates are shown within a record and spaced.

- Metadata are attribute-value pairs m_j, where we do not assume attribute names to be different and we may also have several copies of the same pair; values are of type `string`. We refer to all the attribute-value pairs associated with a given sample s as the `metadata` of s.

A `dataset` is a collection of samples with the same region schema and type; thus, datasets are homogeneous collections of samples, that are typically produced within the same project (either at a genomic research center or within an international consortium) using the same technology and tools. Sample identifiers are unique within each dataset.

In our system we store datasets using two normalized data structures, one for regions and one for metadata; an example of the two data structures for representing a dataset of *ChIP-seq* experiments is shown in Fig. 1. Note that the region value has an attribute of type `real` representing the `p-Value` of each sample region; note also that the `Id` attribute provides a many-to-many connection between regions and metadata of a sample; e.g., sample 1 has 5 regions and 4 attributes, sample 2 has 3 regions and 3 attributes.

2.2 Multi-region genomic entities

The model accommodates also genomic entities which are composed of multiple, correlated regions. Examples are DNA loops, possibly across different chromosomes, chromosome fusions, or multi-specie genome sequences (e.g., viral sequences interleaved with human sequences); these entities have values that allow reconstructing their intra-chromosome regions for specific references through the query language. We formalize loop definitions and chromosome fusions.

- **Loop definitions**[3]**.** The region schema includes the coordinates `chr, left, right, strand` and then the values `read1, read2`; correspond-

[3] These entities are generated by the sequencing of just one terminus of an amplicon per strand, thus generating a couple of mate reads. Mate reads map in different genomic position, but they belong to the same original amplicon. The analysis software fills the gap between

ingly, the first mate goes from `left` to `read1` and the second mate goes from `read2` to `right`, while the loop goes from `read1` to `read2`. When the loop is multi-chromosome, the chromosome `chr` is set to a fictious value (which stands for multi-chromosome regions) and the values `chr1` and `chr2` are added; then the mates go from `left` to `read1` in `chr1` and from `read2` to `right` in `chr2`.

- **Chromosome Fusions**[4]. The region schema includes the coordinates `chr, left, right, strand`, the chromosome `chr` is set to a fictious value, and the values are `offset, chr1, chr2`; correspondingly, the fusion goes from `left` to `offset` in `chr1` and from `offset` to `right` in `chr2`.

The loop and fusion models can be mixed, yielding to a generalized fusion model with two mate regions and a fusion point. Moreover, models from multiple genomes can be mixed, by adding properties such as a different reference. These generalized models are managed *region by region* in the query language; from the query language it is possible to designate such regions by using the schema attributes.

3 Genometric Query Language

GMQL is inspired by *Pig Latin* [3], that combines high-level declarative style in the spirit of SQL with the low-level procedural style of map-reduce [8], [9]. Pig Latin programs are compiled into physical plans which are executed over *Hadoop* [4], [10], an open-source map-reduce implementation. Users of Pig Latin specify a sequence of steps where each step specifies only a single, high-level data transformation. Pig Latin users observe that "the step-by-step method of creating programs in Pig Latin is much cleaner and simpler to use than the single block method of SQL" [3].

3.1 General Structure of GMQL

A GMQL query (or program) is expressed as a sequence of GMQL operations with the following structure:

```
<variable> = operation(<parameters>) <variables>
```

Variables stand for GDM datasets, as shown in Fig. 1. Operations apply to one or more **operand variables** and construct one **result variable**. Parameters are specific of each operation.

the two mates, starting from the reference genome sequence, and restores a unique region. Sometimes, in some data type like ChIA-PET, the two mates are left separated and linked together using a group ID attribute. In ChIA-PET, the attribute `Value` indicates the number of reads which support a given mate pair.

[4] Fusions are events in which genomic DNA from two chromosomes, that is normally separated, is adjacent in a single DNA strand. These regions are thus characterized by a single sequence with an internal fusion point.

Parameters of several operations include predicates, used to select and join datasets or their regions; predicates are built by arbitrary Boolean expressions of simple predicates, as it is customary in relational algebra [5].

Predicates $p(s)$ on regions use the attributes in the region's schema; predicates on metadata may use arbitrary names for metadata attributes. Thus, when a predicate on regions uses an illegal attribute name, the query is also illegal; when a predicate on metadata uses an attribute name missing from $m(s)$, $p(s)$ is `unknown`; as in SQL, we use three-value logics for predicate evaluation and accept in the result of operations only datasets s for which $p(s)$ is `true`. Please note that when evaluating predicates on region attributes that include number costants, the system performs an automatic type casting (that is, conversion) of each attribute according to values contained in the predicate itself rather those stored in the attribute. For instance, when the predicate *value* $>$ *2* is evaluated in GMQL, the values of the attribute *value* are automatically converted to integers before being confronted with 2. This might create loss of precision and undesired output: it is advised to always using double notation for number constants, in our case *value* $>$ *2.0*.

GMQL variables can be prefixed by a clause which redefines its regions:

```
(regions [chr as <attribute name>,] left as <attribute name>,
right as <attribute name> [, strand as <attribute name>])
```

This allows for managing multi-region genomic entities.

We next describe the GMQL operations; they form a closed algebra, hence operation results are expressed as new datasets derived from their operands and from the operation's specifications. For each operation, we provide a informal syntax, where optionality is denoted by square brackets, alternatives by the `|` symbol, and repetition by listing several elements [6]; nonterminal symbols are enclosed between $<$ and $>$; symbols `Si` denote datasets; `si` denote samples; `Ai` denote attributes; `pi` denote predicates; `fi` denote tuple expressions [7]; and `gi` denote aggregate expressions [8]. In JOIN operations, attributes are renamed using the format `<variable name>.<metadata name>` whenever there is ambiguity (as they are homonyms). We also provide a few examples together with the operator description, while many examples are described in the final subsection.

[5] The syntax of simple predicates is `<expression> <comparison> <expression>` where `<expression>` is any parenthesized expression with conventional math operators, constants, and attribute names, and `<comparison>` is one of `<, <=, >, >=, ==, !=` on numeric data, one of `==.  !=` on strings; predicates are build by Boolean parenthesized expressions of simple predicates and the constants `TRUE` and `FALSE`.

[6] With the notation `P1..Pn` we express an arbitrary number of repetitions of `P`.

[7] Tuple expressions are arithmetic expressions of attributes and constants.

[8] Aggregate expressions are arithmetic expressions of aggregate functions over attributes and constants; functions `MIN, MAX, SUM` and `AVG` are applicable to numeric attributes as customary; `EXISTS` is applicable to any attribute and returns 1 if the attribute has some present values, 0 otherwise; `BAG` is applicable to string attributes and builds a record of their values as a comma-separated string; `COUNT` has no argument and counts a number of regions, in a vay that is specific to the operation.

3.2 Operators for Metadata Management

Operations on metadata apply to a single dataset and have the following general features:

- `SELECT` filters out some samples by using a predicate upon metadata attributes.
- `AGGREGATE` applies aggregate functions to region values of each sample and adds the results to new metadata attributes.
- `ORDER` uses metadata attributes for ordering samples and possibly for filtering the top ones based upon the ordering.

All the above operations create a new dataset; the samples which are not filtered out by operations are included in the result dataset without changing their identifiers and their regions; metadata change according to the operation's semantics.

3.2.1 Metadata Selector

The SELECT command selects the samples which satisfy given predicates. Given a dataset `S1`, it applies a predicate p to the metadata $m(s)$ of each sample s of `S1`, and returns into `S2` those samples such that $p(s)$ is satisfied; the symbol `*` can be used equivalently to `TRUE`. In our current implementation, all datasets must be initially selected, as this operation causes the conversion/loading of the corresponding files.

```
<S2> = SELECT(<p>) <S1>
```

An example of select is the following:

```
S2 = SELECT(type == 'gene' AND
           (provider == 'UCSC' OR provider == 'RefSeq')) S1;
```

3.2.2 Metadata Aggregator for Region Values

AGGREGATE computes aggregate functions over region values of each sample and adds the result as new metadata attributes. Given a dataset `S1` and samples s in `S1`, let gi be aggregate expressions over the attributes of the region schema and let Ai be new samples attribute names. For each sample s, the new attribute-value pair $< Ai, gi(s) >$ is added to $m(s)$, where $gi(s)$ is obtained by applying gi to the bag of values vi of attribute Ai in all the regions of s:

```
<S2> = AGGREGATE(<A1> AS <g1>, ..<An> AS <gn>) <S1>
```

For instance, the following function computes the difference between the minimum and maximum `pValue` of the regions of each sample s in `S1` and adds to $m(s)$ the new attribute `pValueOffset`:

```
S2 = AGGREGATE(pValueOffset AS (MAX(pValue) -  MIN(pValue))) S1;
```

3.2.3 Metadata Orderer and Top-k Extractor

The ORDER command orders the samples in ascending or descending order of their metadata, and optionally filters the top samples according to such ordering. Given a dataset S1 and the samples s of S1, let O denote a sequence of n clauses Ai or DESC Ai, where Ai are attributes of $m(s)$; the ascending order is used as default, and DESC denotes the descending ordering. The operator produces a total order of the samples s of S1 which is coherent with the partial ordering induced by O, and adds to the metadata a new attribute Order which explicitly represents such total order. The operation is performed by first attempting the cast of attributes in O from string to a numeric type, and then using the numeric ordering if the cast succeeds, and otherwise the alphanumeric ordering; if a metadata attribute is missing in a given sample, then the sample belongs to the last equivalence class according to the ascending order. Further, the optional clause TOP k filters the first k samples; as an alternative option, TOPG k filters the first k samples for each group as defined by the first $n - 1$ clauses;

```
<S2> = ORDER([DESC]<A1>,..[DESC]<An>[; TOP <k> |; TOPG <k>]) <S1>
```

For instance, the following operation orders samples by sex in ascending order and by weight in descending order within samples with the same value of sex, and then selects the first five samples for each sex.

```
S2 = ORDER(sex, DESC weight; TOPG 5) S1;
```

3.3 Operators for Region Management

Operations for region management apply to regions and have the following general features:

- PROJECT filters out some regions by using a predicate upon region attributes and possibly computes new region attributes or changes the value of existing region attributes.

- COVER and SUMMIT (and their variants) generate new samples whose regions are obtained by taking intersections of operand dataset regions in a variety of ways that depend on operations' parameters.

The PROJECT operation with at least one region satisfying the predicate and the DISTINCT variants of COVER and SUMMIT produce exactly one sample in the result for each sample of their operand; the associated identifier and metadata are not changed, with the exception that a new standard attribute RegionCount is added to the sample metadata, whose value represents the number of regions in the sample. The application of COVER and SUMMIT without DISTINCT variant produces a dataset with a single sample, whose identifier is generated and whose metadata are recomputed.

3.3.1 Region Filter

The PROJECT command filters regions that satisfy given predicates. For each sample s in a dataset S1, an (optional) predicate p is applied to the regions (ri, vi) of s. If no region of s satisfies the predicate, then s does not produce any sample in the result S2; otherwise, s produces a sample of S2 having only the regions where $p(ri, vi)$ is true; the metadata of s are not changed. This implies that if the operand dataset $S1$ has n samples, the result $S2$ will have between 0 and n samples. In addition, existing attributes of filtered regions can be modified or new attributes can be created by using tuple expressions fi.
The syntax is:

```
<S2> = PROJECT([<p>;][<A1> AS <f1>,..
       <An> AS <fn>]) <S1>
```

The names start and stop can be used, only in the project operation, to denote the ends of stranded regions. If the strand is positive or omitted, start applies to the left coordinate and stop applies to the right coordinate (e.g. start = start - 1000 moves the left coordinate thousand bases back in the reference genome). If the strand is negative, start applies to the right coordinate and stop to the left coordinate, and additions/subtractions of coordinates are applied inversely (e.g. start = start - 10 moves the right coordinate of ten bases forward). The operation excludes the regions whose length becomes negative or null (i.e., if the left end goes to the right of the right end.)

The next example shows how a region is extended of 25 bases (in the case of positive or missing strand, the left coordinate is moved 20 bases backward and the right coordinate is moved 5 bases forward; length is computed after changing the right and left coordinates.)

```
S2 = PROJECT(pValue < 0.4 AND chr == 1; start = start - 20,
      stop = stop + 5, length = right - left) S1;
```

3.3.2 Region Cover

COVER extracts a single sample in the result dataset from all samples of the operand dataset; regions of the result sample are formed as combination of regions of the operand dataset, according to the COVER parameters used. Although the operation has intuitive semantics when each sample in the operand(s) contains non overlapping regions, we next define it in the general case of samples with overlapping regions in one or more individual samples. Syntactically, the operation may optionally have two datasets; when a second dataset is present, the result regions must be formed with the mandatory contribution of at least one region of the second dataset. The syntax allows for several optional parts:

```
<S3> = COVER[_FLAT](<minAcc>,<maxAcc>[; JACCARD <minJ>,<maxJ>]
       [; <A1> AS <g1>, ..<An> AS <gn>]
       [; GROUP BY <A1>,..<An>]) <S1> [<S2>]
```

Resulting regions of the result sample are non-overlapping and are built from the regions of samples in S1 (and, when provided, in S2) complying with the following conditions:

a. Each resulting region is the contiguous intersection of at least minAcc and at most maxAcc regions in S1 (and, when provided, in S2), where regions of different strands are separately considered and unstranded regions contribute to both strands; minAcc and maxAcc are called **accumulation indexes**. The most liberal condition corresponds to setting the accumulation indexes respectively to 1 and ANY (see next). When the _FLAT option is used, the operation returns instead the union of all the regions which contribute to the cover. More precisely, the flat option returns a contiguous region that starts from the first end and stops at the last end of the regions which contribute to the cover. Note that with the _FLAT option the result sample may have overlapping regions.

b. When the **Jaccard indexes** minJ, maxJ are provided, resulting regions must in addition have their Jaccard index included within the interval minJ..maxJ; the Jaccard index intuitively measures the degree of overlapping. Omitting the parameters minJ and maxJ is equivalent to setting them to 0 and 1 respectively.

c. If S2 is specified, then at least one region of each sample of S2 must contribute to the regions of the result; regions of S2 are called **mandatory regions**).

It is possible to use the following keywords instead of natural numbers as values for minAcc and maxAcc:

- ANY. It can be used only as maxAcc, and in this case no maximum is set. It is equivalent to omitting the maxAcc option.

- ALL. It is equal to the total number of samples of the operand(s), and can be used both for minAcc and maxAcc; these can also be expressed as arithmetic expressions built by using ALL (e.g. ALL-3, ALL+2, ALL/2; the latter one returns the ceiling (upper integer value) of the division); cases when *maxAcc* is greater than ALL are relevant with overlapping regions.

When the regions in each sample of S1 are non-overlapping, COVER(1,ANY) S1 extracts the union of regions of the samples in S1, COVER(ALL,ALL) S1 extracts the intersections of the regions present in all the samples of S1.

All region values of the original regions are discarded, but the result regions contain as new standard attributes JaccardIndex, a value in the range (0.0 - 1.0] which represents the ratio between the region resulting from the operation and the union of the original regions contributing to such result; thus, the JaccardIndex of COVER / SUMMIT with minAcc, maxAcc parameters set to (1, ANY) is always 1.0, also when the _FLAT option is used. In addition, it is possible to include in the regions new attributes Ai, calculated by appying aggregate functions gi to the original attributes of all the regions of S1 contributing to the result. The COUNT aggregate function counts the number of original regions which contribute to create a region of the result. For instance, the following COVER operation produces output regions where at least 3 regions of S1 overlap, having as resulting region attributes their max pVal, the bag of their rName and the JaccardIndex:

```
S3 = COVER(3, ANY; pVal AS MAX(pVal), names AS BAG(rName)) S1;
```

If we want to impose in addition that the sample S2 (e.g. an annotation) contribute to resulting regions of the COVER, the operation becomes:

```
S3 = COVER(4, ANY; pVal AS MAX(pVal), names AS BAG(rName)) S1 S2;
```

The resulting sample has as metadata the union of the metadata of its input samples in S1, i.e. those attribute-value pairs which appear in any sample of S1. (and, when provided, in S2). In addition, metadata of the result sample have a new attribute-value pair expressing the GlobalJaccardIndex).

The GROUP BY option is useful when cover should be independently computed upon partitions of the datasets which share the same metadata. In this case, each cover is applied to the subset of samples that share the same value for all attributes of a group-by list; if a sample does not have an attribute-value pair for any attribute of the list, it is discarded. The result includes one sample for each equivalence class of the partition; metadata are given by the attribute-value pairs which appear in all the samples of the equivalence class; by construction, they include the attributes of the group-by list. In the next example, a cover is performed on samples that have the same values of antibody and cell:

```
S3 = COVER(4, ANY; GROUPBY 'antibody','cell') S1;
```

3.3.3 Region Summit

SUMMIT is a variant of COVER; it uses the same syntax and produces the same metadata, but differs in the above mentioned condition (a.), as it returns only those portions of COVER results where the maximum number of regions intersect. More precisely, SUMMIT returns regions that start from a position where the number of intersecting regions are not locally increasing afterwards, and stops at a position where either the number of intersecting regions decreases, or it violates the max accumulation index. The syntax is:

```
<S3> = SUMMIT[_FLAT](<minAcc>,<maxAcc>[; JACCARD <minJ>,<maxJ>]
        [; <A1> AS <g1>, ..<An> AS <gn>]
        [; GROUP BY <A1>,..<An>]) <S1> [<S2>]
```

The _FLAT option returns the union of all the regions which contribute to the SUMMIT, rather than the region with the maximum number of intersections. More precisely, _FLAT returns a region that starts from the first end and stops at the last end of the contributing regions. Note that the result sample may have overlapping regions.

3.4 Operations on Multiple Datasets

Operations on multiple datasets have the following features:

- `UNION` applies to many datasets and builds the union of their samples and the merge of their schemas. Metadata of each sample are maintained unchanged.
- `DIFFERENCE` applies to two datasets and preserves the regions of the first dataset which do not intersect with regions of the second dataset. Metadata of the first dataset are maintained unchanged.
- `JOIN` applies to two datasets and pairs both regions and samples of the operand variables when they satisfy metadata and/or region predicates.
- `MAP` applies to two datasets and uses the regions of one single sample dataset as basis for aggregating the properties of each sample of the other dataset (we informally say that the second dataset is *mapped* over the first one.)

3.4.1 Union

UNION takes as input several datasets; the result contains all the samples of its operands, with their original regions and metadata. The schema of the result is equal to the schema of the first operand; attributes of the other operands which are not present in such schema do not contribute to the result. The `UNION` syntax is:

```
<Sn+1> = UNION <S1>..<Sn>
```

3.4.2 Difference

DIFFERENCE takes as input two datasets; for each sample of the first dataset, the operation produces in the result those regions that have no intersection with any of the regions of the second dataset. If the first input dataset has n samples and the second has m samples, the DIFFERENCE result dataset will contain between n and $n-m$ samples. If after such operation a sample has no remaining regions, it is discarded with its metadata; else, metadata of samples of the first operand are unchanged. The `DIFFERENCE` syntax is:

```
<S3> = DIFFERENCE([<meta-join-pred>]) <S1> <S2>
```

An optional meta join predicate is used to associate to each sample of the first dataset a (possibly empty) set of samples of the other datasets; in such case, the difference occurs between each sample of first dataset and the union of samples of the other datasets that satisfy the join condition. Please note that, unlike UNION, a DIFFERENCE command with no meta-join predicate requires empty paretheses, such as in the following example:

```
DIFF = DIFFERENCE() EXPERIMENT UNDESIRED_DATA;
```

3.4.3 Genometric Join

The JOIN command joins the samples (both regions and metadata) of two datasets. The join acts in two phases.

a. First, an (optional) **metadata join predicate**, built as a conjunction of predicates over metadata attributes, produces new samples. If no join metadata predicate is specified, this phase produces as new samples the cross product between the samples of the two datasets.

b. Then, a mandatory **genometric join predicate** is computed over all the pairs of regions of new samples; the genometric join predicate deals with distances of the regions along the reference genome and is expressed in a way that guarantees upper bounds on such distances.

The syntax of join is:

```
<S3> = JOIN[_STRANDED] ([<meta-join-pred>,] <genometric-pred>,
      <region-constr>) <S1> <S2>
```

Given two datasets `S1` and `S2`, let $s1$ denote the samples of `S1` and $s2$ denote the samples of `S2`. The join metadata predicate is built as conjunction of simple predicates with syntax `<Att1> <comparison> <Att2>` (with `Att1` and `Att2` metadata attributes of `S1` or `S2`, respectively); each simple predicate evaluates to `true` when $s1$ has a pair `<Att1,V1>`, $s2$ has a pair `<Att2, V2>`, and the comparison predicate applied to `V1` and `V2` is `true`; it evaluates to `false` otherwise. In this way, pairs $< si, sj >$ of samples are selected from the original datasets; when no join metadata predicate is present, all the pairs in the Cartesian product of `S1` and `S2` are selected. If dataset S1 has n samples and dataset $S2$ has m, the JOIN product will have at most $n * m$ samples, according to the various meta-join and genometric distance conditions imposed. The syntax of meta-join predicates includes a a syntactic disambiguation of attributes for denoting the left and right operands. For instance:

```
left -> antibody == right -> antibody
```

A genometric join predicate is then applied to such pairs; if the predicate is `true`, a new sample s_{ij} is produced, which belongs to the resulting dataset `S3`, and is associated with a new sample identifier.

We first discuss the structure of resulting samples s_{ij}, and then the syntax and semantics of genometric join predicates. Let us assume that the genometric join predicate, applied to regions ri of si and rj of sj, is true. Then:

- New regions r_{ij} are computed by applying the constructor `<region-constr>` to the regions ri of si and rj of sj; the constructor has four options:

  a. `LEFT` returns the left region (i.e. the region ri from the $s1$ sample);

  b. `RIGHT` returns the right region (i.e. the region rj from the $s2$ sample);

c. `INT` returns the region intersection (i.e. the common bases of ri and rj); if the intersection is empty no region is produced by the pair $< ri, rj >$. If no regions are produced for a given sample s_{ij}, then the sample s_{ij} is not created;

d. `CAT`, the concatenation of ri and rj (i.e., all the bases from the lower left end to the upper right end of ri and rj).

Note that several regions with the same coordinates may be produced (either because individual input samples have overlapping regions, or because the multiple regions with the same coordinates are due to genometric predicates); the clauses `PROJECT_` and `_DISTINCT` allow to control these aspects.

a. When the `PROJECT_LEFT` clause is specified, regions of s_{ij} are the same as the regions of the left sample si; we say that the join result is **projected on the left operand.**

b. When the `PROJECT_RIGHT` clause is specified, regions of s_{ij} are the same as the regions of the right sample sj; we say that the join result is **projected on the right operand.**

c. When the clause`_DISTINCT` is added after the constructor `LEFT`, overlapping regions that have the same coordinates and in addition have identical values for all the attributes of the left value v_i are merged. For what concerns the attributes of sj, they are transformed by first casting them to string and then producing a new string value equal to the tab delimited concatenation of such strings.

d. When the clause `_DISTINCT` is added after the constructor `RIGTH`, overlapping regions with same coordinates and identical values for all the attributes of the right value v_j are merged, and attributes of s_i are obtained as the tab delimited concatenation of their values after casting them to string.

e. When the `PROJECT_LEFT_DISTINCT` (or `PROJECT_RIGHT_DISTINCT`) option is used, all regions with the same coordinates and values of s_i (or s_j) are merged, independently of their origin.

- When none of the above clauses is present, new regions are constructed as follows:

  a. The schema of regions r_{ij} is obtained as composition of the schemas of si and sj; it includes the new identifier, the coordinates of the new regions, and then the concatenation of the attributes of the schemas of si and sj other than their identifier and region coordinates. To avoid ambiguity, whenever two region attributes have the same name, the name in the resulting regions contains as prefix the name of the original variable (for example, `S1.Pvalue` and `S2.Pvalue`).

  b. The value v_{ij} of region s_{ij} is obtained by composing the values v_i and v_j of the samples si and sj.

c. Additionally, an attribute `distance` is added to the resulting schema of regions r_{ij} to keep the distance value between the r_i and r_j regions.

- For what concerns metadata, when no `PROJECT_` clause is specified, they are obtained as the set of all the value pairs of si and sj; with the `PROJECT_` clause, metadata of the result are either the same as the metadata of si (option `LEFT` or of sj (option `RIGHT`).

We next turn to genometric join predicates. They are based on the **genometric distance**, defined as the number of bases between the closest ends of two regions. Note that with our choice of interbase coordinates, intersecting regions have distance less than 0 and adjacent regions have distance equal to 0; if two regions belong to different chromosomes, their distance is undefined (and predicates based on distance fail). Genometric predicates are not symmetric; we distinguish between the region of the left variable, also named **reference** or **anchor** region, and the region of the right variable.

Genometric join predicates are composed from the following simple predicates:

a. `DISTANCE <op> <C>`, where `<op>` is either '<' or '>', and `<C>` is a constant. The predicate is true if two regions are at a distance that is either less or greater than `C`; the distance is evaluated between the closest ends of the two regions, one of $s1$ and the other of $s2$. Note that the genometric join predicate `DISTANCE > K` can only be used in the case of an interval condition and only following a `DISTANCE < H` condition (which produces an upper bound).

   We also support `UPSTREAM_DISTANCE` and `DOWNSTREAM_DISTANCE`; they take as reference the anchor region; the former is computed from the anchor region's `start`, that is equal to the `left` end with a positive strand and to the `right` end with a negative strand; the latter refers to the region's `stop`; the distance is positive when it refers to the bases outside the anchor region. When only one of the two predicates is specified, it implicitly refers to the portion of the genome which is either upstream (before the start) or downstream (after the stop) of the anchor region. When both predicates are specified, in disjunctive form, each one refers to one of the two portions of the genome with respect to the anchor region. Distances are computed starting from the region's ends, and by default they are assumed to be positive, (i.e. thus, regions which intersect with the anchor region do not qualify).

b. `OVERLAPPING` is `true` if rj overlaps with the anchor region ri.

c. `MINDISTANCE` is `true` if rj is at minimal distance from ri (the anchor region); more than one region may be returned when several of them are at the same distance. Note that the minimal distance is considered regardless of the up- or down-stream direction from the anchor region. See next simple predicate for specifically focused up- or down-stream evaluations.

d. `FIRST AFTER [UPSTREAM_|DOWSTREAM_]DISTANCE <X>` is a special predicate that is `true` if rj is the closest region farther than X bases from the anchor region ri; more than one region may be returned when several of them are at the same distance.

We further impose that any conjunction of predicates includes either a predicate of class (b), or (c), or (d), or a distance predicate of class (a) with `<op> = '<'` along both strands; the latter one can be in conjunction with a predicate of any other class, or with another class (a) predicate with `<op> = '>'`, with proper `C` values to define a bound interval; a class (a) predicate with `<op> = '>'` can also be in conjunction with a predicate of class (c); additionally, a class (b) predicate can be in conjunction with a predicate of any other class. These conditions guarantee a reasonable bound on the number of regions that satisfy the genometric join predicate.

Thus, for each anchor region the following predicates are legal:

a. `(DISTANCE < 100) AND (DISTANCE > 0)` is `true` for the regions which fall outside of the anchor region without being adjacent to it and are within 100 bases from both its ends.

b. `(DISTANCE < 100)` is `true` for the regions which intersect with the anchor region and are within 100 bases from both its ends.

c. `(UPSTREAM_DISTANCE < 100)` is `true` for the regions in the portion of the genome that fall outside of the anchor region and are within 100 bases before the start of the anchor region.

d. `(UPSTREAM_DISTANCE < 100) OR OVERLAPPING` is `true` for the regions in the portion of the genome between the stop and 100 bases upstream from the start of the anchor region.

e. `(UPSTREAM_DISTANCE < 100) OR (DOWNSTREAM_DISTANCE < 10)` is `true` for the regions in the portion of the genome between the start and 100 bases upstream from the start or between the end and 10 bases downstream from the end of the anchor region.

f. `MINDISTANCE AND (DISTANCE > 100)` is `true` for the closest region(s) to the anchor provided that its distance is greater than 100 bases (thus, this predicate fails if the closest region is at distance less than or equal to 100 bases).

g. `(FIRST AFTER DISTANCE 100)` is `true` for the closest region(s) to the anchor whose distance is greater than 100 bases.

h. `(FIRST AFTER DISTANCE 100) AND (DISTANCE < 1000)` is `true` for the closest region(s) to the anchor whose distance is greater than 100 and less than 1000 bases.

i. `(UPSTREAM_DISTANCE > 100) AND MINDISTANCE` is `true` for the region(s) at minimal distance in the upstream portion from the anchor region, provided that their distance is greater than 100 bases from the anchor.

Note that the predicate `(DISTANCE > 100) AND (UPSTREAM_DISTANCE < 1000)` is not legal, as downstream regions at arbitrary distances may satisfy the predicate. Similarly for `(DISTANCE > 100 AND (DOWNSTREAM_DISTANCE < 1000)`.

The `STRANDED` option is used to indicate that the genometric join predicate should be computed only between regions that belong to the same strand (be it positive or negative; undefined strands should be compared both with positive and negative as well as undefined strands).

3.4.4 Map

The MAP command maps each individual sample of a dataset, called **operand**, to the regions of a specific dataset, called **reference**. We initially discuss the map operation without an optional join extension. In such case, the mapping is performed by first computing the union of the samples of the reference, yielding to the result regions, and then by computing, for each sample of the operand, aggregate functions applied to the non-empty intersections of the result regions with the sample regions; we say that the operand is mapped to the reference. This operation builds a new **genome space** whose regions are the reference's regions and whose samples contain aggregate properties of the original samples relative to the new regions.

Let $s1$ be the union of the reference samples within the dataset `S1` and consider its regions $< ri >$. Let $s2$ be the samples of `S2` and let rj denote their regions. Let Ak be attribute names and let gk be aggregate expressions. Let Di be the multiset of values of the rj region such that ri and rj have non-empty intersection, and let $gk(Di)$ be the result of applying the aggregate expressiom gk to each such multiset Di. Then, the `MAP` operator builds a new dataset `S3` such that:

- Every sample $s2$ of `S2` generates a sample $s3$ of `S3`, with the same identifier and metadata; the metadata of $s3$ include the union of the metadata of $s1$ and of $s2$.
- Regions of `S3` have the same coordinates as regions ri of $s1$;
- Region attributes `Ak` of `S3` corresponding to region coordinate ri take the value $gk(Di)$.

In particulare, if the operand dataset $S2$ has n samples, the resulting mapped dataset $S3$ will have exactly n samples, irrespective of the number of samples in S. The syntax is:

```
<S3> = MAP[_STRANDED] ([<meta-join-pred>,]
       <A1> AS <g1>,.. <An> AS <gn>) <S1> <S2>
```

The aggregate COUNT counts, for each reference region, the number of intersecting operand regions. An example of use of map is:

```
S = MAP(PeakCount AS COUNT) ANNOTATION PEAKS;
```

Assuming that relevant annotations are expressed as regions of a sample within the ANNOTATION dataset, the MAP operation counts the peak regions that are present in the intersections between the annotation regions and the regions of the samples of the PEAKS dataset. Instead, the example:

```
S = MAP(Peak AS EXISTS) ANNOTATION PEAKS;
```

Returns 1 in the annotation regions with at least one peak, and 0 otherwise.

An optional <meta-join-predicate> is used for selecting pairs of samples of the first and second operand based on their metadata; in such case, the map operation is applied to the pairs $< S_i, S_j >$ such that samples S_i of S1 and S_j of S2 satisfy the join predicate. For instance:

```
S = MAP(left -> antibody == right -> antibody,
     PeakCount AS COUNT) CONTROL PEAK;
```

The _STRANDED option is used to indicate that the intersecting multi-set should include only regions which belong to the same strand (be it positive or negative; undefined strands should be compared both with positive and negative strands).

References

[1] ENCODE Project Consortium, Bernstein BE, Birney E, Dunham I, Green ED, Gunter C, Snyder M. An integrated encyclopedia of DNA elements in the human genome. Nature. 2012;489(7414):57-74.

[2] Kent WJ, Sugnet CW, Furey TS, Roskin KM, Pringle TH, Zahler AM, Haussler D. The human genome browser at UCSC. Genome Res. 2002;12:996-1006.

[3] Olston C, Reed B, Srivastava U, Kumar R, Tomkins A. Pig Latin: A not-so-foreign language for data processing. In: Proceedings ACM SIGMOD 2008, NY: ACM; 1099-1110.

[4] Shvachko K, Kuang H, Radia S, Chansler R. The Hadoop distributed file system. Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST). IEEE Computer Society, Washington, DC, 2010. 1-10.

[5] Data File Formats [http://genome.ucsc.edu/FAQ/FAQformat.html]

[6] Dowell RD, Jokerst RM, Day A, Eddy SR, Stein L. The distributed annotation system. BMC Bioinformatics 2001;2:7.

[7] Nicol JW, Helt GA, Blanchard SG Jr, Raja A, Loraine AE. The Integrated Genome Browser: free software for distribution and exploration of genome-scale datasets. Bioinformatics 2009;25(20):2730-27301.

[8] Ekanayake J, Pallickara S, Fox G. MapReduce for data intensive scientific analyses. IEEE Fourth International Conference on eScience (eScience '08). IEEE Computer Society, Washington, DC, 2008. 277-284.

[9] Zou Q, Li XB, Jiang WR, Lin ZY, Li GL, Chen K. Survey of MapReduce frame operation in bioinformatics. Brief Bioinform. Feb 2013.

[10] Taylor RC. An overview of the Hadoop MapReduce HBase framework and its current applications in bioinformatics. BMC Bioinformatics 2010;11 Suppl 12:S1.

[11] Prosad PJ, Bodhe GL. Trends in laboratory information system. Chemom Intell Lab Syst 2012, 118:187-192.

[12] Eisen MB, Spellman PT, Brown PO, Botstein D. Cluster analysis and display of genome-wide expression patterns. Proc. Natl. Acad. Sci. USA 1998;95(25):14863-14868.

[13] Müller H, Mancuso F. Identification and analysis of co-occurrence networks with NetCutter. PLoS One. 2008;3(9):e3178.

[14] Nordberg H, Bhatia K, Wang K, Wang Z. BioPig: a Hadoop-based analytic toolkit for large-scale sequence data. Bioinformatics 2013;29(23):3014-3019.

[15] Cock PJ, Fields CJ, Goto N, Heuer ML, Rice PM. The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. Nucleic Acids Res. 2010;38(6):1767-1771.

[16] Li H, Handsaker B, Wysoker A, Fennell T, Ruan J, Homer N, Marth G, Abecasis G, Durbin R; 1000 Genome Project Data Processing Subgroup. The Sequence Alignment/Map format and SAMtools. Bioinformatics 2009;25(16):2078-2079.

[17] Fernndez-Surez XM, Galperin MY. The 2013 Nucleic Acids Research Database Issue and the online molecular biology database collection. Nucleic Acids Res. 2013, 41(Database issue):D1-D7.

[18] Meyer LR, et. Al. The UCSC Genome Browser database: extensions and updates 2013. Nucleic Acids Res. 2013;41(Database issue):D64- D69.

[19] Röhm U, Blakeley J. Data management for high-throughput genomics. CoRR 2009, arXiv:0909.1764 [cs.DB]. Asilomar, CA, USA; ACM. 2009. 1-10.

[20] Bafna V, Deutsch A, Heiberg A, Kozanitis C, Ohno-Machado L, Varghese G: Abstractions for genomics. ACM Communications 2013, 56(1):83-93.

[21] Cereda M, Sironi M, Cavalleri M, Pozzoli U. GeCo++: a C++ library for genomic features computation and annotation in the presence of variants. Bioinformatics 2011, 27(9):1313-1315.

[22] Ovaska K, Lyly L, Sahu B, Jänne OA, Hautaniemi S. Genomic Region Operation Kit for flexible processing of deep sequencing data. IEEE/ACM Trans Comput Biol Bioinform. 2013;10(1):200-206.

[23] Quinlan AR, Hall IM. BEDTools: a flexible suite of utilities for comparing genomic features. Bioinformatics 2010;26(6):841842.