Jacquard Documentation

Release 0.42

Jessica Bene, Ashwini Bhasi, Chris Gates, Divya Kriti, Kevin Meng

Oct 30, 2018

Contents

CHAPTER 1

Overview

Jacquard is an open source suite of Python command line tools that provides a practical approach to integrating multiple patient samples and multiple variant callers. Jacquard is designed to be used by bioinformatic analysts; the output is intended to be useful to analysts and biological researchers. Both Jacquard and its documentation assume that users the basics of variant callers and VCF files. For more information about VCF files, see the Hts-specs.

1.1 Why would I use Jacquard?

Jacquard makes it easier to analyze multi-patient tumor-normal datasets especially when using multiple vairant callers.

Most variant callers have embraced the Variant Call Format (VCF) standard *[r2]*, a file format which clearly and succinctly describes variants from one or more samples. However, while many callers follow the standard, they often adopt different ways to partition results (e.g. somatic file vs. germline file, or SNP vs. indel); likewise, each caller creates its own dialect of VCF fields and tags *[r3] [r5] [r7]*.

Moreover, each variant caller follows its own algorithms, and produces different results for the same inputs. Because of this, it is valuable to run data through multiple variant callers and compare the outputs *[r3] [r5] [r7]*. However, since each caller has its own dialect, direct comparisons are difficult.

Jacquard transforms the dialects of different variant callers into a controlled vocabulary of tags with a consistent representation of values. Furthermore, it intelligently merges VCFs from different patients and callers to create a single, unified VCF across your dataset. The consistent tag names and represntations expedite downstream analysis; the intgrated VCF highlights both the prevelance of specific variants and the overall mutation loads across samples.

Jacquard can merge or expand VCFs from any variant caller. Jacquard can translate depth, alt frequency, somatic status, and genotype tags from several somatic variant callers:

- MuTect *[r1]*
- VarScan *[r4]*
- Strelka *[r6]*

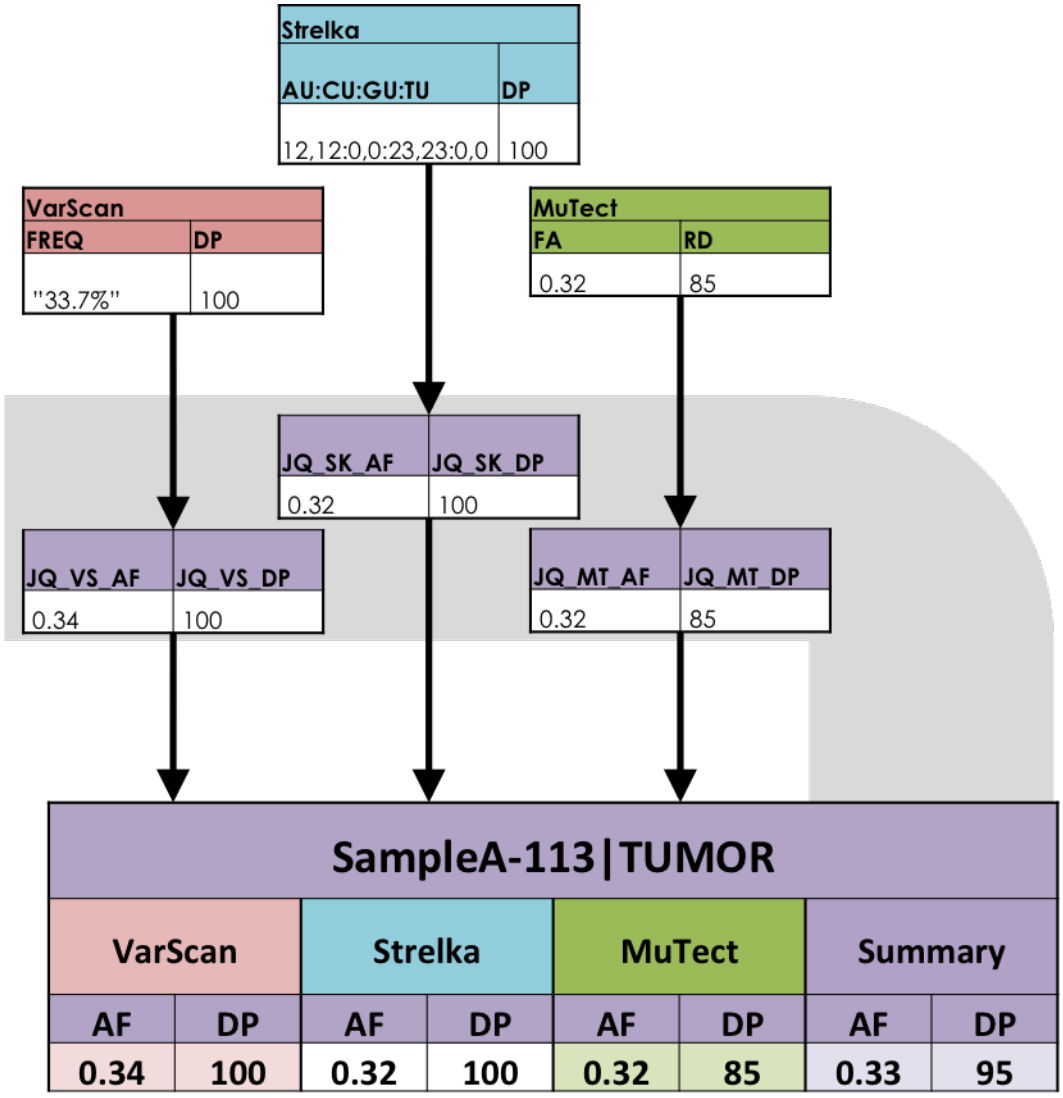

Fig. 1: **Jacquard normalized VCF dialects:** *Each variant caller records depth or alt frequency with a different tag name and representation. Jacquard translates format tags from different callers into a uniform set of tags.*

1.2 Contact Us

Email bfx-jacquard@umich.edu for support and questions.

UM BRCF Bioinformatics Core

CHAPTER 2

Installing Jaquard

Jacquard has been tested with Python 2.7 and 3.4 on Windows7, OSX, and *nix.

2.1 Prerequisites

Note: Pip installs all required libraries; see [Installing] below.

- natsort (3.5.2)
- nosetests, testfixtures (3.0.2), and numpy (>=1.7.1) are required for running automated tests

2.2 Installing

The easiest way to install Jacquard is through PyPI. Get pip if it's not available in your system:

```
$ pip install jacquard
```

You can install from source from github:

```
$ pip install git+https://github.com/umich-brcf-bioinf/Jacquard
```

If you don't have root permissions, you can install locally:

```
$ pip install --user jacquard
```

Note: You may need to modify your path to include the Python install dir (e.g. /Users/<username>/.local/bin)

CHAPTER 3

Quick Start

This is a simple tutorial on how to use the four Jacquard commands.

1. **Install Jacquard (see *Installing Jaquard*).**

2. **Unzip the `examples.zip` file to your home directory (or other directory of your choice).**

   The examples directory contains sample input VCFs from five patients run with three variant callers. The input VCFs are based on a subset of actual variant calls from clinical data; the samples were de-identified and VCF positions have been randomized to prevent downstream identification. As a result of randomization, the sample VCF reference calls at a specific position don't always match the base calls from a reference sequence.

   Along with the inputs, the example dierctory contains output from each Jacquard command, as explained below.

3. **Create an output directory.**

4. **The *translate* command creates new VCFs, adding a controlled vocabulary of new FORMAT tags.**

```
$ jacquard translate examples/00-input_vcfs/ <output_dir>
```

5. **The *merge* command integrates a directory of VCFs into a single VCF.**

```
$ jacquard merge examples/01-translated/ <output_vcf_file>
```

6. **The *summarize* command adds new INFO fields and FORMAT tags that combine variant data from the merged VCF.**

```
$ jacquard summarize examples/02-merged.vcf <output_vcf_file>
```

7. **The *expand* command explodes a VCF file into a tab-delimited file.**

```
$ jacquard expand examples/03-summarized.vcf <output_tsv_file>
```

Refer to *Overview* for more information on Jacquard.

CHAPTER 4

Command Details

Jacquard is a suite of tools that can be either run in succession or individually; the typical workflow is to run:

4.1 Translate

The *translate* command creates new VCFs, adding a controlled vocabulary of new FORMAT tags. It will only work with VCF files from the supported variant callers.

4.1.1 Usage

```
$ jacquard translate <input_dir> <output_dir> [OPTIONS]
```

positional arguments:

input_dir	Directory containing VCF files (and VarScan high confidence files)
output_dir	Directory containing VCF files. Will create if doesn't exist and will overwrite files in output directory if –force

optional arguments:

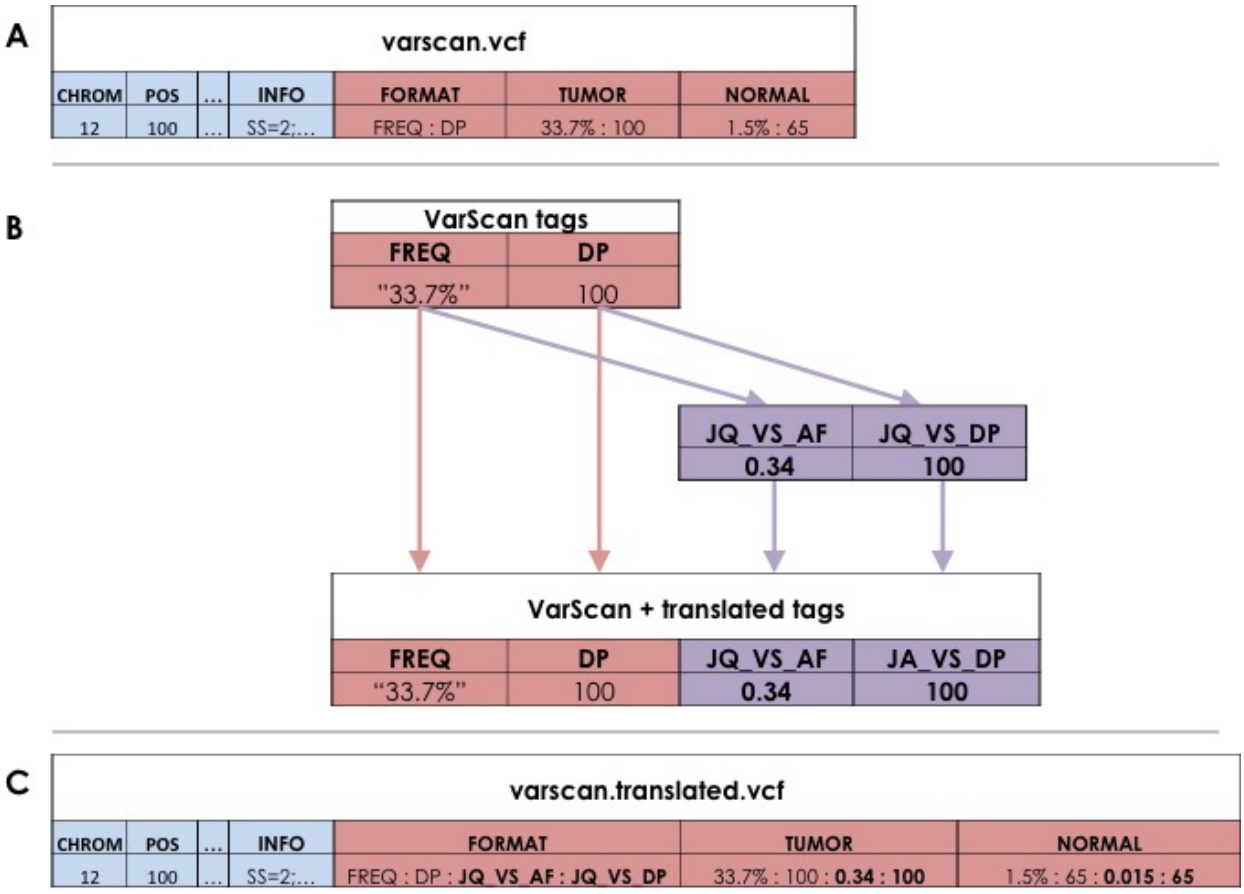

Fig. 1: **Translate adds new FORMAT Tags :** *(A) Jacquard reads input VCF files (B) deriving new tags based on information in each variant record and (C) writes new VCFs appending the new tags to the original record.*

–allow_inconsistent_sample_sets	Set this flag if not every patient is represented by the same set of caller-VCFs. (For example if you ran VarScan on only a subset of cases.)
–varscan_hc_filter_file_regex=	Regex pattern that identifies optional optional VarScan high-confidence filter files. The VCF, high-confidence file pairs should share the same prefix. For example, given files: patientA.snp.vcf patientA.indel.vcf patientA.snp.fpfilter.pass patientA.indel.fpfilter.pass you could enable this option as varscan_hc_filter_file_regex= '.fpfilter.pass$'

4.1.2 Description

The *translate* command accepts a directory of VCF files and creates a new directory of "translated" VCF files, which include several Jacquard-specific FORMAT tags and their corresponding metaheaders.

You can either gather all input VCFs into a single directory and run *translate* once or partition VCFs into separate directories (for example, by variant caller) and run *translate* once for each input directory. When partitioning into separate input directories, all file names must be unique.

Currently, *translate* adds Jacquard-specific FORMAT tags for:

- Allele Frequency
- Depth
- Genotype
- Somatic Status
- Passed: Indicates whether the variant record as a whole passed the VC filters; this tag is used later on when merging translated VCFs
- Reported: This tag is used when merging translated VCFs.

See VCF metaheader excepts below for more details on how values are derived:

4.1.3 Strelka Translated Tags

Tag name	Description
JQ_SK_AF	Jacquard allele frequency for Strelka: Decimal allele frequency rounded to 4 digits (based on alt_depth/total_depth. Uses [TIR tier 2]/DP2 if available, otherwise uses (ACGT tier2 depth) / DP2)
JQ_SK_DP	Jacquard depth for Strelka (uses DP2 if available, otherwise uses ACGT tier2 depth)
JQ_VS_GT	Jacquard genotype (based on SGT). Example for snv: ALT=C, INFO SGT=AA->AC is translated as normal=0/0, tumor=0/1. Example for indel: INFO SGT=ref->het is translated as normal=0/0, tumor=0/1.
JQ_VS_HC_SOM	Jacquard somatic status for Strelka: 0=non-somatic,1=somatic (based on PASS in FILTER column)

4.1.4 MuTect Translated Tags

Tag name	Description
JQ_SK_AF	Jacquard allele frequency for MuTect: Decimal allele frequency rounded to 4 digits (based on FA).
JQ_SK_DP	Jacquard depth for MuTect (based on DP)
JQ_MT_GT	Jacquard genotype (based on GT)
JQ_MT_HC_SOM	Jacquard somatic status for MuTect: 0=non-somatic,1=somatic (based on SS FORMAT tag)

4.1.5 VarScan Translated Tags

Tag name	Description
JQ_VS_AF	Jacquard allele frequency for VarScan: Decimal allele frequency rounded to 4 digits (based on FREQ)
JQ_VS_DP	Jacquard depth for VarScan (based on DP)
JQ_VS_GT	Jacquard genotype (based on GT)
JQ_VS_HC_SOM	Jacquard somatic status for VarScan: 0=non-somatic, 1=somatic (based on SOMATIC info tag where sample column is TUMOR and variant record passed VarScan filter).

Jacquard can incorporate VarScan high-confidence files

To run *translate* with VarScan calls, Jacquard requires the VarScan VCF files (snp and/or indel). For each VarScan VCF, Jacquard can optionally accept VarScan somatic high-confidence files; these are supplemental non-VCF files that list variant records which passed a more stringent set of VarScan filters.

When high-confidence files are present, the *translate* command adds a FILTER field value for low-confidence variant records (i.e. records which may have initially passed filters, but are absent in the high-confidence files).

To use VarScan's somatic high-confidence files, they must be placed alongside corresponding VarScan VCFs and must have the same file name prefix as their corresponding VCF file. The high-confidence filename suffix can be specified using the command line argument.

Example VarScan files:

case_A.varscan.indel.vcf
case_A.varscan.indel.Somatic.hc.filter.pass
case_A.varscan.snp.vcf
case_A.varscan.snp.Somatic.hc.filter.pass
case_B.varscan.indel.vcf
case_B.varscan.indel.Somatic.hc.filter.pass
...

4.2 Merge

The *merge* command integrates a directory of VCFs into a single VCF. It is caller-agnostic and can be used on any set of VCF files.

4.2.1 Usage

```
$ jacquard merge <input_dir> <output_file>
                 [--include_format_tags=JQ_.*]
                 [--include_cells=valid]
                 [--include_rows=at_least_one_somatic]
```

positional arguments:

input_dir	Directory containing input VCF files to be merged
output_file	An integrated VCF file

optional arguments:

–include_format_tags=	Comma-separated user-defined list of regular expressions for format tags to be included in output; (defaults to **'JQ_.*'**)
–include_cells=	all: Include all variants **valid**: Only include valid variants passed: Only include variants which passed their respective filter somatic: Only include somatic variants
–include_rows=	all: Include all variants at loci at_least_one_passed: Include all variants at loci where at least one variant passed all_passed: Include all variants at loci where all variants passed **at_least_one_somatic**: Include all variants at loci where at least one variant was high-confidence somatic all_somatic: Include all variants at loci where all variants were high-confidence somatic
–include_all	Equivalent to: –include_format_tags='.*' –include_cells=all –include_rows=all Useful when merging untranslated VCFs.

4.2.2 Description

Conceptually, *merge* has four basic steps, each described in detail below.

1. Integrate matching loci from different VCFs into common rows
2. Combine matching samples from different VCFs into common columns
3. Filter tag values and rows
4. Assemble the subset of FORMAT tags to be included in the final VCF

Integrate matching loci

Merge first develops the superset of all loci (CHROM, POS, REF, and ALT) across the set of all input VCFs. For each locus, the input VCF FORMAT tags and values are merged into a single row. Input variant record-level fields (such as

FILTER, INFO, etc.) are ignored.

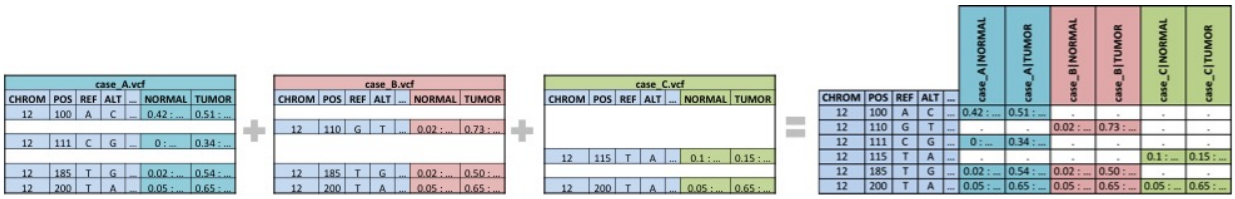

Fig. 2: **Matching loci :** *Variant records from separate files that share the same CHROM, POS, REF, ALT are merged into a single variant record.*

Combine matching samples

In the input directory, an individual sample could be called by more than one variant caller. When merging, Jacquard combines results from the same sample into a single column. Merged sample names are constructed by concatenating the filename prefix and the VCF column header.

Filename	VCF Column header	Merged sample names
case_A.strelka.vcf	#CHROM ... FORMAT SAMPLE1 SAMPLE2	case_A|SAMPLE1 case_A|SAMPLE2
case_A.mutect.vcf	#CHROM ... FORMAT SAMPLE1 SAMPLE2	case_A|SAMPLE1 case_A|SAMPLE2
case_B.strelka.vcf	#CHROM ... FORMAT SAMPLE3 SAMPLE4	case_B|SAMPLE3 case_B|SAMPLE4
case_B.mutect.vcf	#CHROM ... FORMAT SAMPLE3 SAMPLE4	case_B|SAMPLE3 case_B|SAMPLE4

Given the input VCFs above, the resulting merged VCF will have four sample columns:

- case_A | SAMPLE1
- case_A | SAMPLE2
- case_B | SAMPLE3
- case_B | SAMPLE4

Filter cell values and rows

Variant records are filtered to highlight the high-confidence somatic variants.

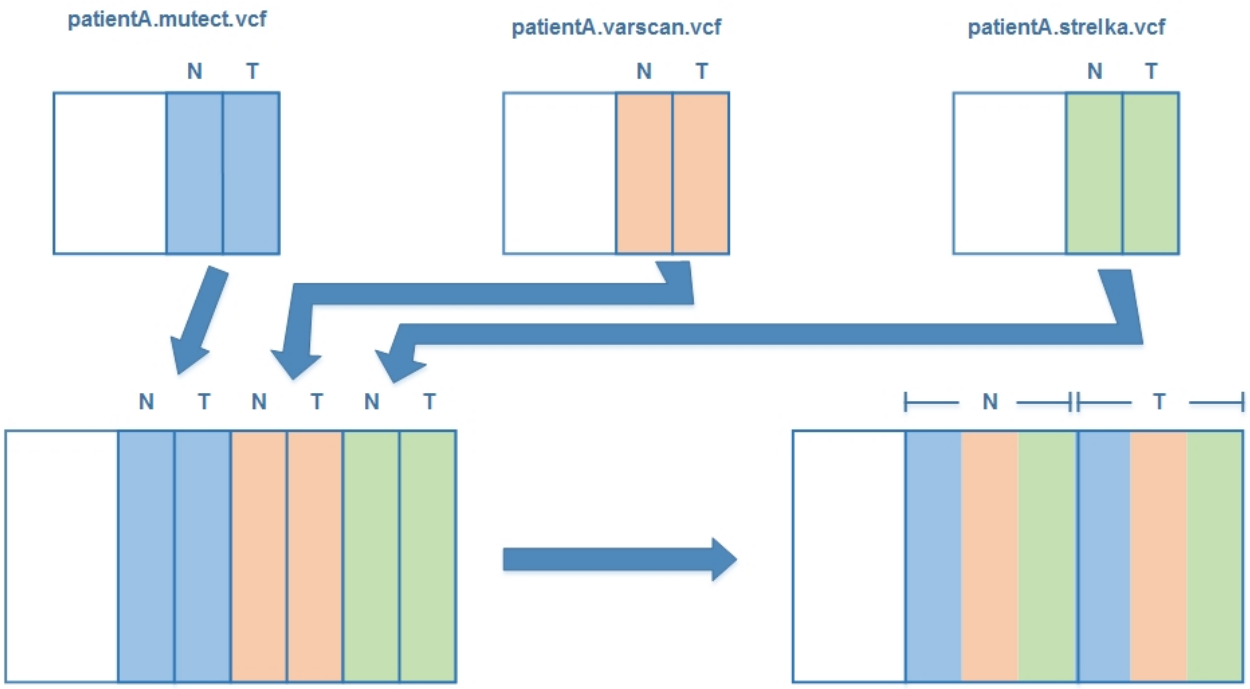

Fig. 3: **Combine matching samples :** *Case-specific information reported in different files is combined into a single case\|sample column.*

For VCFs from supported callers, *merge* filters the result to include only valid variants records where at least one variant at that loci was somatic. The filter stringency can be set with flags described above. Since these filters operate on Jacquard tags, *merge* cannot filter VCFs from unsupported callers; use –include_all for untranslated VCF files.

Assemble the subset of FORMAT tags

Merge builds a new set of INFO tags and returns a subset of incoming FORMAT tags. By default, Jacquard only carries forward tags that begin with 'JQ', i.e. Jacquard-translated tags. When working with VCFs from unsupported callers, use –include_format_tags or –include_all to merge unstranslated VCFs.

Note that while most variant callers have their own distinct set of FORMAT tags, some tag names are common across multiple callers. If there are any FORMAT tag name collisions, *merge* will add a prefix (e.g. JQ1_<original_tag>) in order to disambiguate the FORMAT tags.

4.3 Summarize

The *summarize* command adds new INFO fields and FORMAT tags that combine variant data from the merged VCF. It will only work with VCF files that have been translated.

4.3.1 Usage

```
$ jacquard summarize <input_file> <output_file>
```

positional arguments:

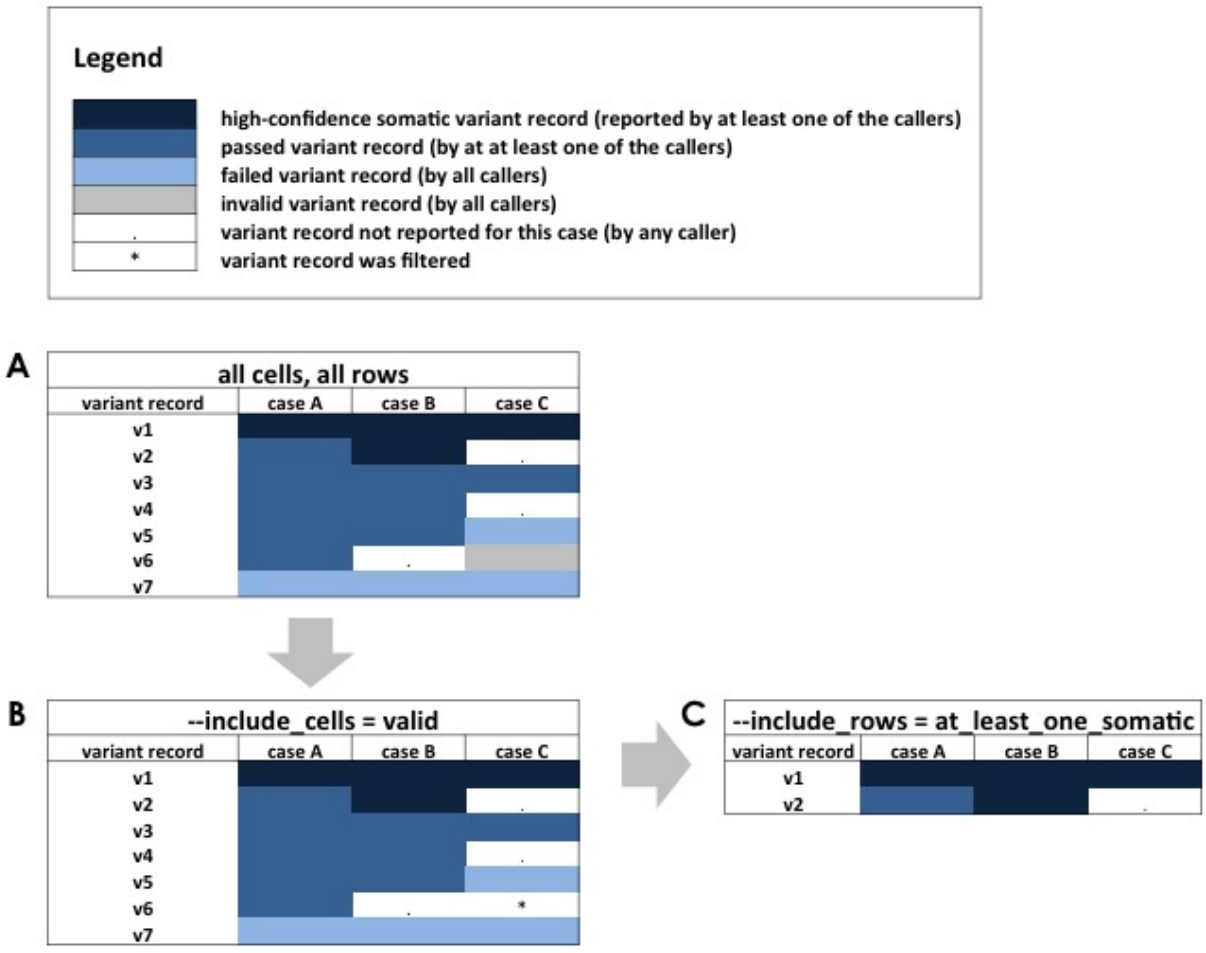

Fig. 4: **Filter cell values and rows :** *(A) Beginning with the matrix of all variant records, (B) the include_cells flag transforms excluded cells (sample-records) from their original value to "." (not-observed). (C) Finally, the include_row flag excludes entire loci.*

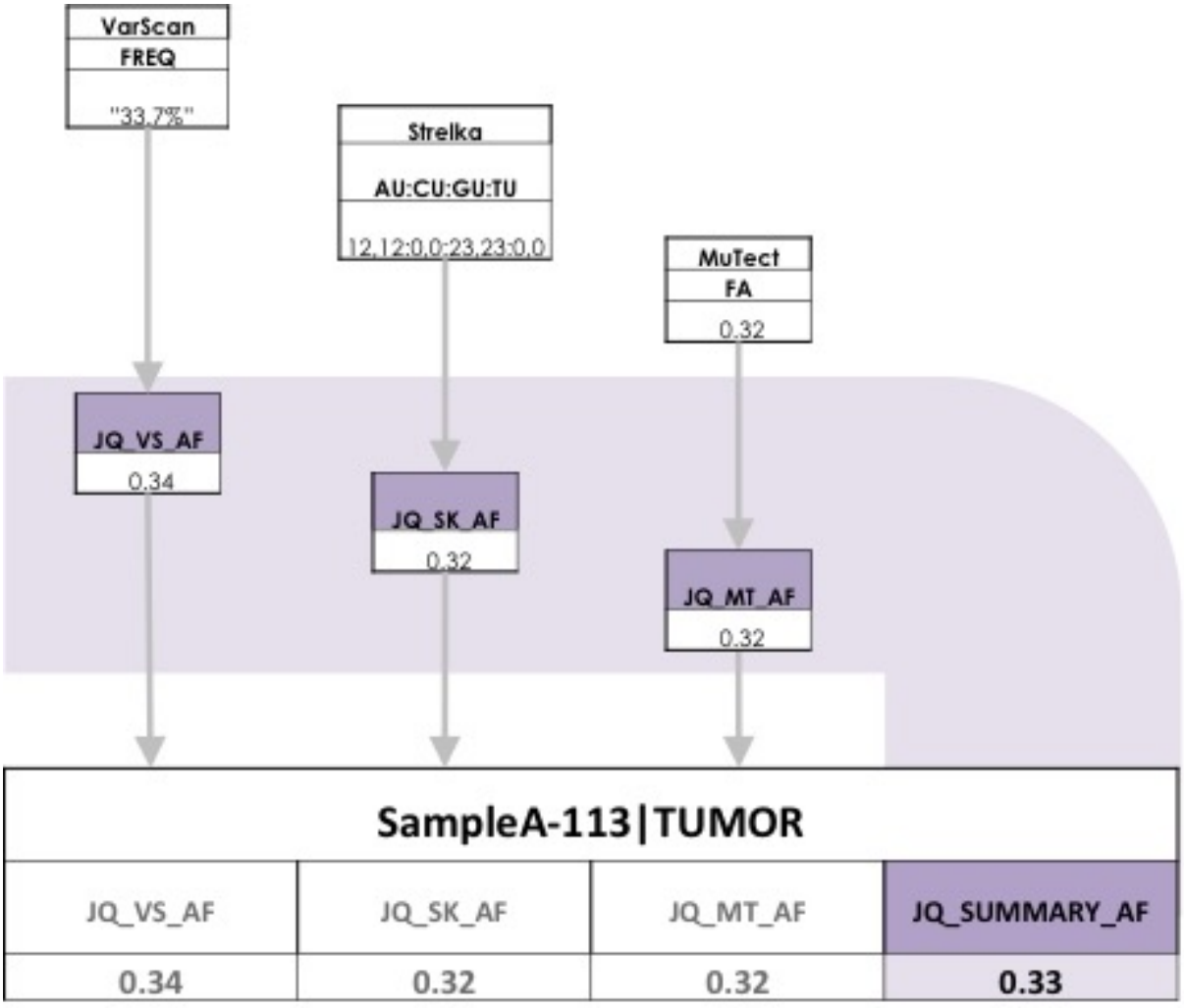

Fig. 5: **Summarizing Format Tags :** *The Jacquard-translated format tags from each caller are aggregated to create summary format tags.*

input_file	Jacquard-merged VCF file (or any VCF with Jacquard tags; e.g. JQ_SOM_MT)
output_file	A single VCF file

4.3.2 Description

The *summarize* command uses the Jacquard-specific tags to aggregate caller information from the file, providing a summary-level view. Summary fields (e.g. average allele frequency) can highlight interesting variants.

The summarized format tags contain the prefix 'JQ_SUMMARY'.

Example summary FORMAT tags

Tag name	Description
JQ_SUMMARY_AF_AVERAGE	Average allele frequency across recognized variant callers that reported frequency for this position [average(JQ_*_AF)].
JQ_SUMMARY_AF_RANGE	Max(allele frequency) - min (allele frequency) across recognized callers.
JQ_SUMMARY_HC_GT	High confidence consensus genotype (inferred from JQ_*_GT and JQ_*_CALLER_PASSED). Majority rules; ties go to the least unusual variant (0/1>0/2>1/1). Variants which failed their filter are ignored.
JQ_SUMMARY_SOM_COUNT	Count of recognized variant callers that reported confident somatic call for this sample-position.

Refer to the summary VCF metaheaders for a full list of summary tags and descriptions.

4.4 Expand

The *expand* command explodes a VCF file into a tab-separated file. It is not caller-dependent and will work with any VCF file.

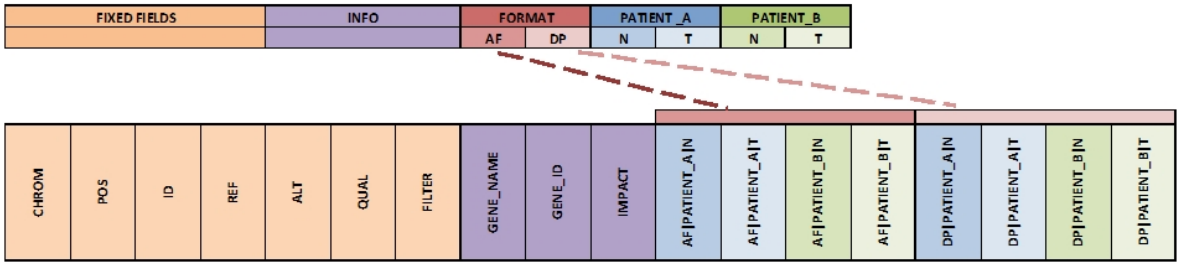

Fig. 6: **Expanding Columns :** *The INFO column and sample-specific FORMAT tags from the input VCF file are separated into distinct columns in the output file.*

4.4.1 Usage

```
$ jacquard expand <input_file> <output_file> [OPTIONS]
```

positional arguments:

input_file	A VCF file
output_file	A tab separated text file

optional arguments:

-s, –selected_columns_file FILE	File containing an ordered list of column names to be included in the output file; column names can include regular expressions

4.4.2 Description

Expand command converts a VCF file into a tab-delimited file. This format is more suitable than a VCF for analysis and visualization in R, Pandas, Excel, or another third-party application.

4.4.3 Note

- The 'fixed' fields (i.e. CHROM, POS, ID, REF, ALT, QUAL, FILTER) are directly copied from the input VCF file.
- Based on the metaheaders, each field in the INFO column is expanded into a separate column named after its tag ID.

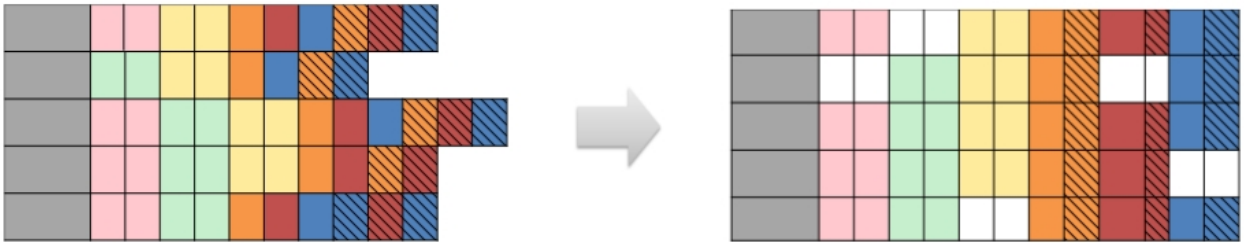

Fig. 7: **Tabular format of expand output:** *Expand transforms the dense VCF format into a tabular format.*

- Each FORMAT tag is expanded into a set of columns, one for each sample, named as <FORMAT tag ID>|<sample column name>.
- By default, all INFO fields and FORMAT tags are expanded; specific INFO fields and FORMAT tags can be selected using the –selected_columns_file option.
- *Expand* also emits a tab-delimited glossary file, based on the metaheaders in the input VCF file. FORMAT and INFO tag IDs are listed in the glossary and are defined by their metaheader description.

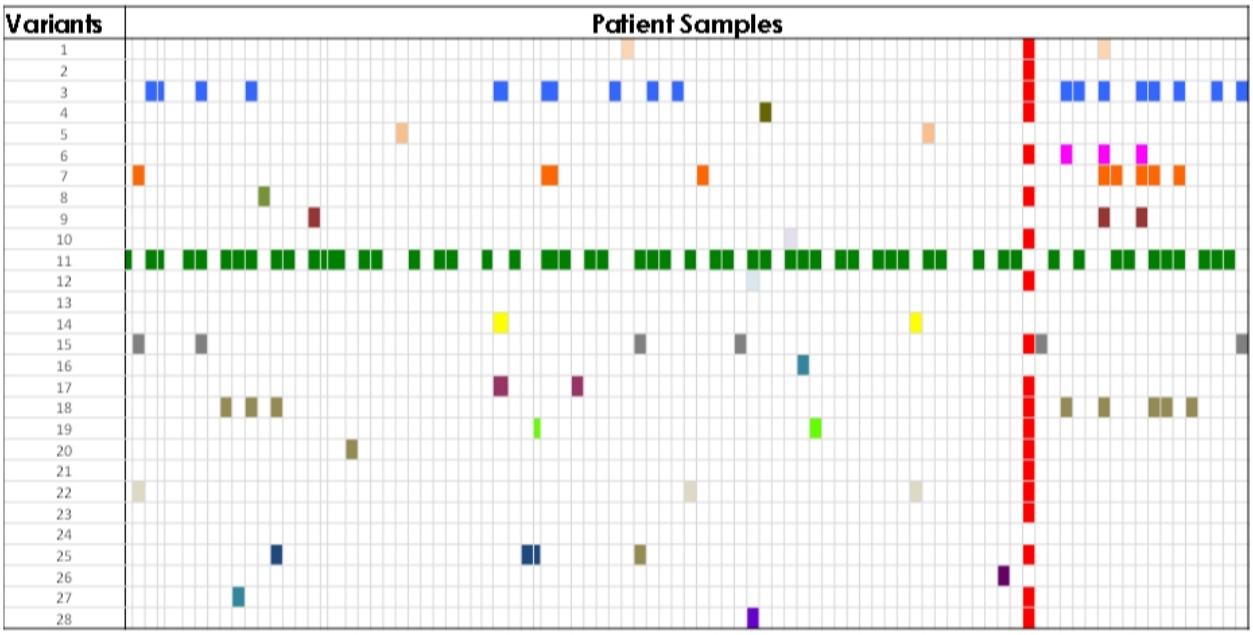

Fig. 8: **Pattern Identification :** *The expanded output file can be visualized in a third-party tool to identify patterns in the dataset.*

Translate and *summarize* commands are useful only for supported callers; *merge* and *expand* work for any VCFs. Each of these commands is described in detail in the following pages.

General usage

```
$ jacquard <SUBCOMMAND> [ARGUMENTS] [OPTIONS]
```

For help on a specific command:

```
$ jacquard <SUBCOMMAND> --help
```

- Jacquard first writes output files to a temporary directory and only copies the files upon successful completion of each subcommand.
- Error, warning, and info messages are written to console and log file. Debug messages are only written to the log file (unless –verbose specified).

Input File Conventions

- Jacquard assumes that the first element of the filename (up to the first dot) is a patient identifier. For example:
  - patientA-113.mutect.vcf
  - patientA-113.strelka.snv.vcf
  - patientA-113.strelka.indel.vcf

  This set of three files all have the same patient identifier (patientA-113). The tumor-normal sample pairs will be combined into a single pair of tumor-normals columns in the merged VCF. See *merge* for more details.
- To translate a specific VCF dialect, Jacquard determines the source variant caller based on the VCF metaheaders. For this reason it is essential that you preserve all metaheaders in the source VCF.
- For a specific source VCF, Jacquard automatically determines the tumor and normal samples based on the column header and the metaheaders.

CHAPTER 5

Frequently Asked Questions

Is Jacquard a variant caller? Jacquard is not a variant caller. It accepts VCF output from variant callers and integrates them for simplified annotation and analysis.

Can Jacquard annotate data? No, Jacquard cannot annotate data; however the output from *translate*, *merge*, and *summarize* can be run through an annotation tool such as SnpEff or Annovar.

Can I use Jacquard with any variant caller? *Merge* and *expand* are able to process VCF files from any variant caller. *Translate* and *summarize*, however, must be run with VCF files from one or more of the supported variant callers. Currently, Jacquard supports MuTect, VarScan, and Strelka.

I'd like to merge my VCFs, but my caller isn't supported by Jacquard. Both *merge* and *expand* commands can be used to show all of the results from different callers without standardization of the input data. However, it is recommended that the input data be standardized whenever possible to directly compare data across callers.

Does Jacquard work with germline callers? The *translate* command is optimized to work with tumor-normal sample pairs. Germline VCFs can be used with *merge* and *expand* commands. Better support for germline and pedigree VCFs is coming soon.

5.1 Still Have Questions?

Email bfx-jacquard@umich.edu for support and questions.

CHAPTER 6

Changelog

6.1 1.1.1 (10/30/2018)

- Adjusted Mutect translators to:
  - parse normal and tumor designations from SAMPLE metaheaders if available
  - recognize more variations of Mutect metaheader formats

6.2 1.1.0 (6/18/2018)

- Adjusted *translate* to correctly parse newer versions of Mutect
- Updated supported versions for Mutect, Strelka, Varscan
- Fixed error in JQ_SUMMARY_DP_AVERAGE tag description

6.3 1.0.0 (6/5/2018)

- Removed obsolete spikes directory
- Fixed bug in *expand* which could overwrite fixed VCF fields (e.g. REF, ALT, etc.) if identically named fields in INFO.
- Switched to semantic versioning

6.4 0.42 (9/22/2015)

- Added docs on readthedocs.
- Improved workflow documentation with example data

- *Merge* will now disambiguate tag collisions from multiple VCs
- *Translate*/*summarize* now supports GT tags
- Extended precision to 4 decimal places to support analysis of gene-panels.
- Adjusted translate to handle empty high-confidence VarScan files.

6.5 0.41 (5/7/2015)

- Combined *filter* command with *merge* command
- Extended *expand* to create simple metaheader glossary
- Adjusted code to support Python >=2.7 or 3.x
- Improved checks for consistent VCF file sets
- Fixed bug in *merge* that caused error if any VCFs were unsorted
- Fixed bug in *summarize* that caused error if variant was called by subset of callers

6.6 0.31 (3/17/2015)

- Downgraded VCF format from 4.2 to 4.1
- Fixed a bug that omitted CALLERS_REPORTED_LIST summary tag
- Simplified summary tags; removed dependency on numpy
- Adjusted VarScan translation to accept a file pattern to identify high-confidence files

6.7 0.3 (3/9/2015)

- Replaced *normalize*, *tag* commands with *translate*; relaxed constraints on incoming data.
- Renamed *consensus* to *summarize*
- More consistent behavior in *expand*
- Significantly improved *merge* performance
- Added new summary tags:
  - CALLERS_REPORTED_COUNT
  - CALLERS_REPORTED_LIST
  - SAMPLES_REPORTED_COUNT
  - CALLERS_PASSED_COUNT
  - CALLERS_PASSED_LIST
  - SAMPLES_PASSED_COUNT
- Fixed bug in how Strelka calculated AF on indels
- Improved command validation and error handling
- Added project/code documentation

- Removed dependencies on pandas

6.8 0.21 (10/2014)

- Initial public release

CHAPTER 7

Future Directions

- Parallelize *translate*
- Improve performance of *merge*
- Add *weave* command to combine *translate*, *merge*, *summarize*
- Extend *expand* to parse SnpEff/Annovar annotated results
- Extend *expand* to generate formatted results
- Improve command validation (check source tags, check “shape” of inputs)
- Enable 4.2/4.3 VCF support
- Add support for new somatic callers
- Add support for Germline workflows
- Add support for Galaxy integration
- Add gene-level rollup for annotated data

CHAPTER 8

Implementation details

8.1 Coding Conventions

- Code should support Python 2.7 and 3.x without modification.
- Use pylint to keep code PEP8 compliant.
- Name variables, method, directory, module names as my_thing.
- Name classes as MyClass.
- Use absolute imports.
- Encapsulate where possible; prefix private vars/methods/classes with "_".
- Avoid globals.

8.2 Test Conventions

- Use nosetests for automated testing.
- Each module has a corresponding module_test.
- TestCases should extend JacquardBaseTestCase
- Top level methods are tested in ModuleTestCase.
- Each class should have a test case (e.g. MyClassTestCase)
- Every code path should have a unit test; prefer single assert per test. A test method name should reflect the method and code path as {test_method_name}_{conditionUnderTest}; e.g.

```
def test_my_method_returnsZeroIfMissingInput(self):
```

- Every command should have a functional test
- Prefer unit tests to functional tests

- Prefer tests on public methods, but note that it is sometimes easier to test a private method. Use good judgement.
- Attempt PEP8 compliance.
- Make unit tests independent.

8.3 General Architecture:

Modules are typically one of these:

- commands (like *translate*): these modules are invoked from the command line; they follow a simple command pattern.
- variant caller transforms (like *mutect*): these modules contain classes that add Jacquard annotations to a native VCF record.
- utilities (like *vcf* or *logger*): these modules provide a common method or class used by other modules.

Note that translate is the only command that should understand variant caller dialects; other commands should be caller agnostic.

Extending and adapting existing patterns will ensure commands/transforms stay consistent. Here are some guidelines on how to extend functionality:

8.3.1 How to add a new format tag:

For all variant callers that support the new tag, you will need to extend each variant caller transform to:

- define the new tag (set the metaheader and how the new value is derived); by convention, tags ID values are JQ_<caller_abbreviation>_<tag_name>
- add the new tag to the variant caller's reader

Note: If the new tag can be summarized, you will also need to add a corresponding tag to *summarize_rollup_transform*.

8.3.2 How to add a new variant caller:

- Add a new module in *variant_caller_transforms*.
- In the new module, define the supported version.
- Add supported tags (as described in section above).
- Add a VcfReader class to interpret native VCFs to translated VCFs.
- Add a new class named for the variant caller; define a claim method to recognize and claim VCF files.
- Add the new variant caller class to *variant_caller_factory*.

..note:: The variant caller should have no dependencies on other packages (except utils and vcf) and classes should only refer to variant callers through *variant_caller_factory* (except tests).

8.3.3 How to add a new command:

- Add a new module in *jacquard*, named for the command.
- In the new module, add the methods:
  - add_subparser(subparser) with appropriate help and defaults.
  - get_required_input_output_types().
  - validate_args(args).
  - report_prediction
  - execute(args, execution_context).

Note: Commands are independent and should not refer to other commands.

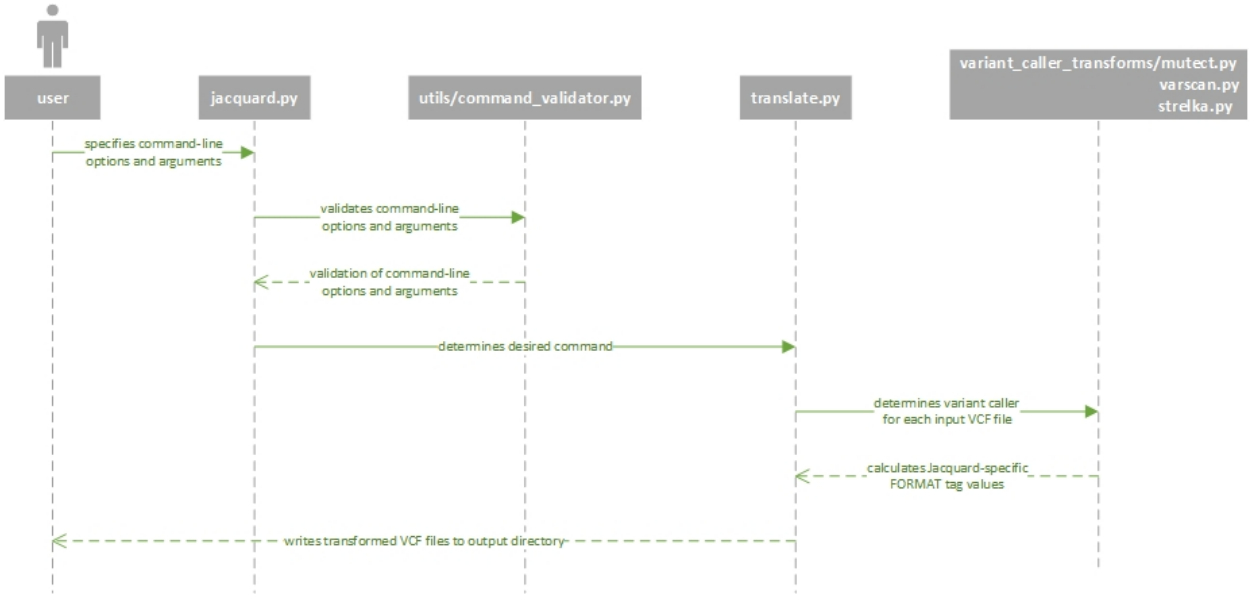

Fig. 1: **UML Sequence Diagram :** *An example UML sequence diagram for Translate. Other commands follow a similar sequence.*

CHAPTER 9

References

CHAPTER 10

License

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

   "License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

   "Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

   "Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

   "You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

   "Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

   "Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

   "Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

   "Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other

modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

   (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.
6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.
7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.
8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.
9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

Bibliography

[r1] Cibulskis K et al. Sensitive detection of somatic point mutations in impure and heterogeneous cancer samples. Nat Biotechnol 2013, 31:213-219.

[r2] Danecek P, Auton A, Abecasis G, Albers CA, Banks E, DePristo MA, et al. The variant call format and VCFtools. Bioinformatics 2011; 27: 2156–8.

[r3] Kim SY, Speed TP. Comparing somatic mutation-callers: beyond Venn diagrams. BMC Bioinformatics 2013, 14:189.

[r4] Koboldt DC et al. VarScan 2: somatic mutation and copy number alteration discovery in cancer by exome sequencing. Genome Res 2012, 22:568-576.

[r5] O'Rawe J et al. Low concordance of multiple variant-calling pipelines: practical implications for exome and genome sequencing. Genome Med 2013, 5:28.

[r6] Saunders CT et al. Strelka: accurate somatic small-variant calling from sequenced tumor-normal sample pairs. Bioinformatics 2012, 28:1811–7.

[r7] Xu H. et al. Comparison of somatic mutation calling methods in amplicon and whole exome sequence data. BMC Genomics 2014, 15: 244.