Sequence analysis

A Python package for parsing, validating, mapping and formatting sequence variants using HGVS nomenclature

Reece K. Hart¹,²,*; Rudolph Rico¹; Emily Hare¹; John Garcia¹; Jody Westbrook¹ and Vincent A. Fusaro¹

¹Invitae Inc., San Francisco, CA 94107 and ²23andMe Inc., Mountain View, CA 94043, USA

Associate Editor: John Hancock

1 INTRODUCTION

As high-throughput sequencing becomes commonplace in the investigation and diagnosis of disease, it is essential that communicating variants from sequencing projects to the scientific community and from diagnostic laboratories to health-care providers is easy and accurate. The Human Genome Variation Society (HGVS) mutation nomenclature recommendations (Taschner and den Dunnen, 2011) are widely endorsed by professional organizations, mandated by numerous journals and displayed by databases and tools. The HGVS recommendations—originally devised to standardize the representation of variants discovered before the advent of high-throughput sequencing—are now approved by the HGVS and continue to evolve under the auspices of the Human Variome Project. The continual evolution of HGVS guidelines makes the nomenclature difficult to understand and to use for experts and non-experts alike, often resulting in incorrect usage and potential clinical interpretation errors.

We sought a software library for manipulating HGVS variants that was appropriate for clinical diagnostics. Specifically, we required the following functionality: (i) We must be able to process patient variants locally (i.e. not sent to a remote site); (ii) we must be able to audit, extend and control updates of the source code and data; (iii) we must be able to map variants in regions of genome-transcript discrepancies, particularly indels. Mutalyzer (Wildeman et al., 2008) provides a Web interface and Web services for constructing, validating and transforming sequence variants, but is not available for local installation. Another Python library (https://github.com/counsyl/hgvs), similar in spirit to the one we present here, uses a regular expression-based parser and relies on exon structures derived from BLAT alignments (Kent, 2002). Although other packages accept or generate HGVS-formatted variants, such as snpEff (Cingolani et al., 2012) and VEP (McLaren et al., 2010), they are not intended for use as a software library.

Here we present an open-source Python library for parsing, mapping, validating and formatting sequence variation according to the HGVS guidelines. The library features a parser based on a Parsing Expression Grammar and a variant mapper that accommodates insertion/deletion discrepancies between reference genomic sequences and transcripts that confound most existing tools.

2 PACKAGE OVERVIEW

The hgvs Python package comprises six key components: (i) object models for representing components of HGVS-formatted variants; (ii) a parser that generates an object representation from an HGVS-formatted string; (iii) formatting tools that generate an HGVS-formatted string from an object representation; (iv) mapping classes that transform variants between genomic, CDS and protein representations; (v) validation tools that ensure conformance to HGVS guidelines; (vi) an interface for defining external data sources required for validation and mapping.

The core of the hgvs package is a set of object models that provide a foundation for developers to reference any component or property of the HGVS syntax. For example, a SequenceVariant consists of a sequence accession, a sequence type and PosEdits, which represent a set of individual changes to a sequence. A PosEdit is composed of positions, such as a BaseOffsetPosition for a CDS (c.) SequenceVariant with intronic variants, and an Edit for Single Nucleotide Variant (SNV), del, ins, delins, duplications and repeats. Figure 1a shows an example

*To whom correspondence should be addressed.
object representation. For a full list of object classes, readers should consult the source code.

The HGVS parser presented here is based on a parsing expression grammar and was inspired by previous work (Laros et al., 2011). There are 127 parsing rules to cover DNA, RNA, CDS and protein parsing. Any rule may be invoked to parse components of valid HGVS strings into an appropriate type. For example, the c_edit rule may be used to parse an

Fig. 1. Using the hgvs package to project a variant in MCL1 from one transcript to another via GRCh37 chromosome 1. (a) An object representation of the result of parsing ‘NM_182763.2:c.688+403C>T’. Selected attributes are shown beneath. (b) A diagram of the MCL1 locus with five representations of a single variant. (c) Python code that demonstrates parsing, mapping between sequences, formatting and validating. Gray outline boxes enclose input, and the results appear immediately beneath. Circled numbers indicate a correspondence between the variants in (a) and code in (c). An SNV, originally reported in literature as NM_182763.2:c.688+403C>T (rs201430561), is projected onto chromosome 1 as variant g, and then projected to an alternative transcript as variant h. The inferred protein changes of variants e and h are shown as protein variants f and i. The results are formatted by ‘stringifying’ them using standard Python printing commands. Validation for a valid variant (281C>T; no error generated) and an error for an invalid variant (281A>T) are shown.

Downloaded from https://academic.oup.com/bioinformatics/article-abstract/31/2/268/2366029 by guest on 16 March 2020
HGVS substring such as ‘c.688+403C>T’. Following Python conventions, formatting is implemented using the str() method of each class.

The hgvs package requires sequence data and exon structures to map variants between the genome and transcript coordinates, to infer protein sequence changes from transcripts and to validate variants. The data provider interface declares seven methods required to support hgvs functionality; developers may implement a subclass of the abstract interface to use data from other sources. The hgvs package includes a concrete implementation based on the publicly available Universal Transcript Archive (information about UTA is available with package documentation).

In addition to the syntactic validation provided during parsing, the hgvs package includes validation tools to ensure that a SequenceVariant object conforms to the HGVS guidelines and to catch common errors. For performance reasons, the Validator distinguishes intrinsic and extrinsic validation: intrinsic validation asserts internal correctness of the object, such as requiring that the start position is less than or equal to the end position or that the length of the location range specified for an insertion is one, and extrinsic validation invokes external data for validation, such as verifying that the reference sequence specified in the variant matches that from a source database.

The hgvs package provides tools to transform (‘map’) variants between genomic (g.), mRNA (r.), CDS (c.) and protein (p.) variants. Of particular note is the implementation of an indel aware mapper that correctly accounts for insertions and deletions in transcripts with respect to a genomic reference. These discrepancies occur owing to natural polymorphisms and sequencing errors, and occur in $\sim$1.6% of current RefSeq transcripts (Garla et al., 2011). The package also includes a transcript liftover tool to migrate variants between different transcripts. When used in conjunction with UTA, liftover may also be performed between the same RefSeq transcript aligned to a genomic reference by Splign (Kapustin et al., 2008) and BLAT (Kent, 2002). Splign and BLAT provide substantially different exon structures for $\sim$2.7% of RefSeq transcripts.

3 TESTING, VALIDATION AND LIMITATIONS

Reliable and robust variant manipulation is an essential goal of this work. The hgvs package implements extensive automated tests that are run on every commit made to the hgvs code, and the test results are publicly accessible. Unit tests, which validate parsing, formatting, mapping and validating, are extensive. In particular, 163 package—parsing, formatting, mapping and validating. A Figure 1 illustrates the five primary functions provided by the hgvs package—parsing, formatting, mapping and validating. A variant in MCL1, NM_182763.2:c.688+403C>T (rs201430561), is parsed into a variant object that exposes variant components as Python properties. The variant is then projected onto NM_001197320.1 via GRCh37 and a splign-based exon alignment. A validation error is demonstrated with NM_001197320.1:c.281A>T, which specifies an incorrect reference nucleotide.

Conflict of interest: All authors are employed by and have equity in the company that sponsored this work.

REFERENCES

Cingolani,P. et al. (2012) A program for annotating and predicting the effects of single nucleotide polymorphisms, SnpEff: SNPs in the genome of Drosophila melanogaster strain w1118; iso-2; iso-3. Fly (Austin), 6, 80–92.
Garla,V. et al. (2011) MU2A—reconciling the genome and transcriptome to determine the effects of base substitutions. Bioinformatics, 27, 416–418.
Green,R.C. et al. (2013) ACMG recommendations for reporting of incidental findings in clinical exome and genome sequencing. Genet. Med., 15, 565–574.
Kapustin,Y. et al. (2008) Splign: algorithms for computing spliced alignments with identification of paralogs. Biol. Direct, 3, 20.
Kent,W.J. (2002) BLAT—the BLAST-like alignment tool. Genome Res., 12, 656–664.
Laros,J.F.J. et al. (2011) A formalized description of the standard human variant nomenclature in Extended Backus-Naur Form. BMC Bioinformatics, 12 (Suppl 4), S5.
McLaren,W. et al. (2010) Deriving the consequences of genomic variants with the Ensembl API and SNP Effect Predictor. Bioinformatics, 26, 1–2.
Taschner,P.E.M. and den Dunnen,J.T. (2011) Describing structural changes by extending HGVS sequence variation nomenclature. Hum. Mutat., 32, 507–511.
Wildeman,M. et al. (2008) Improving sequence variant descriptions in mutation databases and literature using the mutalyzer sequence variation nomenclature checker. Hum. Mutat., 29, 6–13.