Semantic Variation Graphs: Ontologies for Pangenome Graphs

Toshiyuki T. Yokoyama[1], Simon Heumos[2]*, Josiah Seaman[3], Dmytro Trybushnyi[4], Torsten Pook[5], Andrea Guarracino[6], Erik Garrison[7,8] and Jerven T. Bolleman[9]*

[1]*The University of Tokyo, Chiba, Japan*

[2]*Quantitative Biology Center (QBiC) Tübingen, University of Tübingen, Tübingen, Germany*

[3]*Max Planck Institute for Developmental Biology, Tübingen, Germany*

[4]*Karlsruhe Institute of Technology, Karlsruhe, Germany*

[5]*Center of Integrated Breeding Research, University of Goettingen, Goettingen, Germany*

[6]*University of Rome Tor Vergata, Via della Ricerca Scientifica 1, Rome, Italy*

[7]*Genomics Institute, University of California Santa Cruz, Santa Cruz, CA, USA*

[8]*Biomolecular Engineering and Bioinformatics, University of California Santa Cruz, Santa Cruz, CA, USA*

[9]*SIB Swiss Institute of Bioinformatics, Geneva, Switzerland*

** Contributed equally.*

ABSTRACT

Background: Variation graphs are a novel way to describe genomic variation across a population. Variation graph tools present a significant improvement in mitigating reference bias compared to the linear reference ecosystem. Existing toolkits focus on algorithms processing pangenome graphs. Yet, they have limited capabilities in integrating various annotations of the biology and providing an interface for large scale visualizations.

Description: To interpret biological meaning in variation graphs by integrating various kinds of annotations for further analysis, FAIR data interchange formats are demanded. Borderless technology such as the Semantic Web allows variation graph toolkits and pangenome tools to focus on their core competence while allowing bioinformaticians to integrate, analyze, and visualize the data.

Result: We demonstrate how we can represent a graphical pangenome with pangenome ontologies using a standard declarative graph query language. Then we show how the vg RDF and Pantograph RDF can represent data ready for the Semantic Web and how we can combine existing data from INDSC and UniProt without conversions or loss of information into a single Variation and Knowledge Graph.

1 INTRODUCTION

Continuous improvement in sequencing throughput and cost has enabled us to collect population-scale sequence data (Sudmant et al. 2015). The number of sequenced genomes and non-negligible reference bias suggest a pressing need for reference graph genome (Garrison et al. 2018, Ballouz et al. 2019). The Variation Graph toolkit (vg) is one of the pioneering works to provide plenty of graph genome algorithm implementation. As many genomes are integrated into a graph, more and more variations are encoded on it, forming a graphical pangenome[1]. Consequently, the visualization becomes more complicated, demanding an efficient way to visualize them (Yokoyama et al. 2019). Pantograph[2] is handling this challenge by merging nodes as components using the binned, linked nodes provided by odgi (Eizenga et al. 2020). Including Pantograph, continuous effort on graph genome tools are undergoing by many developers around the world.

Interpretation of the variation encoded in the pangenome graph requires integration with existing genomic annotations and subsequent visualization. However, there is no general way to represent pangenome graphs as subject-predicate-object triples, thus hindering efficient integration with annotations and variation graph, and communication in a more general way between tools, e.g. frontend and backend in a web application.

Here we present two ontologies, vg RDF and Pantograph RDF, to be general enough as pangenome ontologies to represent variation graph and pangenomic order of linked nodes. We employ Semantic Web technologies to integrate with various kinds of genomic annotations because they have been adopted as a data exchange format by many life science databases (Bolleman et al. 2016 and Kawashima et al. 2018), making graphical pangenomes FAIRer (Wilkinson et al. 2016 and Ballouz et al. 2019). vg RDF is able to encode variant graph topologies. Pantograph RDF

[1] https://pangenome.github.io/

[2] https://graph-genome.github.io/

encodes bins, components, and links on top of vg RDF to enable linear visualization of large scale data. We describe the definition of vg RDF and Pantograph RDF, and their application of a real-world dataset.

2 IMPLEMENTATION

Both the vg and Pantograph ontologies were implemented in the repository of the variation graph toolkit[3]. An overview of potential application cases, especially with respect to visualization, is given in Fig. 1.

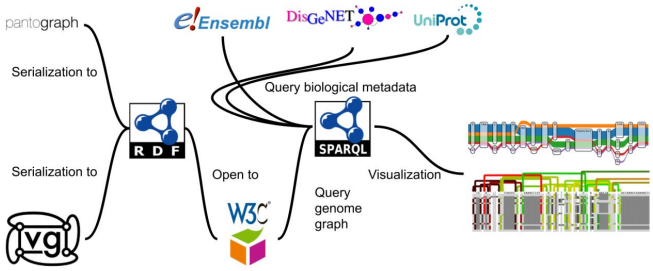

Fig. 1. The schematic diagram of RDF on vg and Pantograph ontology exemplifying the integration of annotation data from different SPARQL endpoints.

2.1 Variation Graph Schema Ontologies

Variation graphs are directed nucleotide sequence graphs and are therefore expressible in RDF. The central concepts in variation graphs are nodes, edges and paths. Sequence is stored in nodes being linked by bidirect edges. Paths through the nodes describe linear genomic sequences.

A Node in the vg RDF is equivalent to a node in the vg data model. The same applies for Edges. A Path is a list of Steps that represent a sequence of Node visits showing its linear biological sequence. Each Step connects a Node into a Path (Fig. 2).

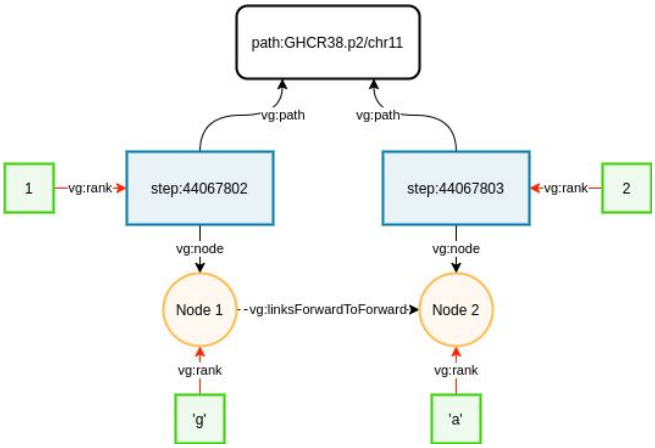

Fig. 2. An example of class and properties used in vg RDF.

[3] https://github.com/vgteam/vg/blob/master/ontology/vg.ttl

Paths may visit the same node more than once, which allows the representation of tandem repeats. The topology of the variation graph is expressed with *vg:Nodes* and *vg:links* and it's subproperties. Moreover, *vg:Paths* and *vg:Steps* are required to reconstruct linear molecular sequence representations (i.e., to make 'FASTA' files).

A key consideration is permanence of node identifiers. Currently these are not tracked between builds. While they could be nodes, for pragmatic reasons we have skolemized them and given them IRIs for which we can not guarantee stability, nor collision. It is technically possible to deploy a variation graph with stable node IRIs given enough resources.

2.2 Pantograph Ontologies

Here the pangenome graph is a subset of a variation graph characterized by a much larger number of genomes, demanding a more effective way to visualize the entire graph. odgi calculates a linear order of nodes called Pangenome Sequence and separates nodes into bins by a certain width called bin width. Non-linear edges caused by structural rearrangement are called links. Links are preserved after linear order is calculated. In the Pantograph pipeline, Component Segmentation groups bins into components and bindles the links to assign on two ends of each component. By specifying the bin width (zooming level), we have components, each of which has one or more bins. Pantograph RDF is a view-oriented ontology inspired by the development of Pantograph.

Pantograph is a small web ontology language version 2 (OWL2) ontology with 7 classes, 1 of these is external, FALDO, and deals with the concept of a position on a Bin (Fig. 3). In a variation graph, nodes can be grouped as bins. To increase the aggregation level, odgi introduces the concept of bins, defined as combinations of nodes into longer DNA segments of a given length (bin width). We have multiple layers of bins with different bin sizes, which we denote as zoom layers. The zoom layer is intended to provide a zoom in/out feature in a visualization tool. Each *vg:ZoomLevel* represents all *components* of a given *zoomFactor* of a pangenome. Each *vg:Component* has one or more *bins* inside. Each *vg:Bin* has one or more cells inside. The concept can be seen as a matrix-like layout, where the x-axis is a list of bins, and the y-axis is a list of paths. *vg:Cell* is an intersection of both axes on the matrix, i.e., a cell represents a subsequence of the path. *Cell* has a reference to the *faldo:position* via *cellRegions* property. *Cell* also has *inversionPercent* (a fraction of inversion) and *positionPercent* (a fraction of the length of a subsequence of the path occupying this cell).

Components and bins are aligned in the order of the Pangenome Sequence. *Forward/reverse edge* and *rank* are

used to indicate the adjacency on the Pangenome Sequence. *Forward/reverse edge* and *rank* work complementary; *rank* is a simple way to represent order, and *edge* is easy for querying adjacent nodes. Users can use either *edge* or *rank* for describing adjacency of components or bins. In addition, there is a *vg:Link*, which is a non-linear edge between bins. Link has one or more *vg:Path* via *linkPaths*. Link also has *arrival* and *departure* bins to represent an arrow between two distant bins. Because the order of link matters in visualization, *forward/reverse edge* and *rank* are stored in each link.

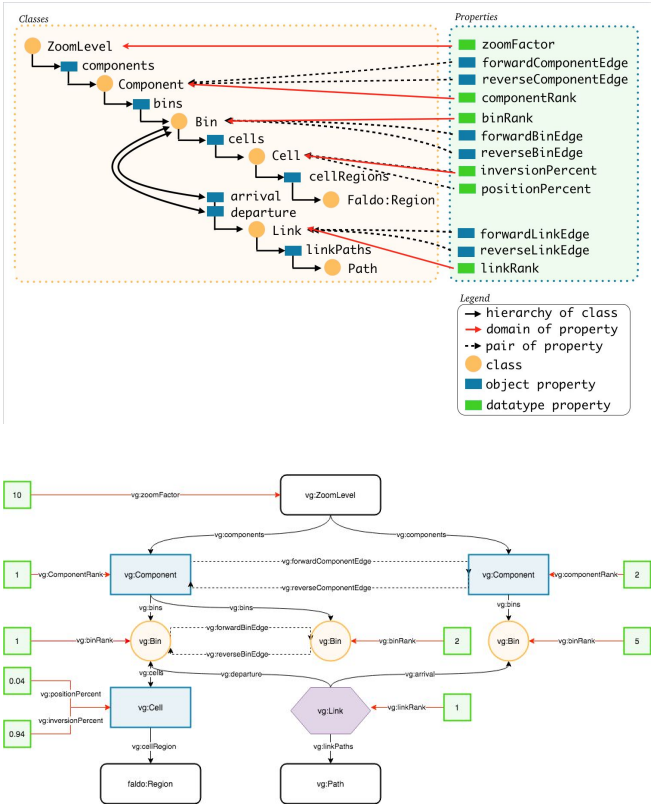

Fig. 3. The class and properties used in Pantograph ontology and its example.

3 RESULT

3.1 SPARQL Querying a Semantic Genome Graph

During Japan Biohackathon 2019 we extended the JavaScript library Sequence Tube Map (Beyer et al. 2019) to display MatrixTubeMaps[4]. As a first test of the vg Ontology, we demonstrated that it is possible to SPARQL query a vg RDF triple store and visualize the response as a MatrixTubeMap[5].

3.2 Variation Graph of an *SARS-CoV-2* Pangenome

During the virtual biohackathon-2020 spodgi is used to translate and seqwish[6]/odgi build variation graph into a linked pangenome. This pangenome of more than 800 viral genomes is loaded into a SPARQL endpoint with the associated INDSC, UniProt and other SIB Swiss Institute of Bioinformatics data in RDF (https://covid-19-sparql.expasy.org) this integrated knowledge base allows us to answer questions such as "are the active sites for PL1-PRO conserved" and what about the surrounding sequence stretch? If not, is there a geographic spread which pinpoints where the mutations happened? Linking to external data resources such as wikidata, for names of regions in multiple languages. An example of how such a SPARQL endpoint could be queried is given in Fig. 4.

```
SELECT
    DISTINCT
        ?insdCDS #?insdCDSBegin ?insdCDSEnd ?step
        ?uniprot ?stepBeginInProteinSpace ?stepEndInProteinSpace ?annotationText
WHERE
{
#  Find CDS annotated by INDSC that do not match a UniProt protein.
  ?insdCDS insdc:translation ?sequence ;
        a insdc:Coding_Sequence ;
        faldo:location ?insdCDSLocation .
  MINUS {
      ?uniprotSequence rdf:value ?sequence .
  }

#   Get the range of this CDS and make sure the coordinates are on the
#  path we need later
  ?insdCDSLocation faldo:begin [ faldo:reference ?path ;
                                 faldo:position ?insdCDSBegin] ;
                   faldo:end [ faldo:reference ?path ;
                               faldo:position ?insdCDSEnd] .

  ?step a vg:Step ;
      vg:path/skos:closeMatch ?path ;
        vg:node ?node ;
      faldo:begin [ faldo:reference/skos:closeMatch ?path ;
                                                    faldo:position ?insdcStepBegin ] ;
      faldo:end [ faldo:reference/skos:closeMatch ?path ;
                                faldo:position ?insdcStepEnd ] .
## I always forget how to interval ranges :(
  FILTER ( (?insdcStepBegin >= ?insdCDSBegin && ?insdcStepBegin <= ?insdCDSEnd) ||
           (?insdCDSBegin >= ?insdcStepBegin && ?insdCDSBegin <= ?insdcStepEnd) ||
                    (?insdcStepEnd >= ?insdCDSBegin && ?insdcStepEnd <= ?insdCDSBegin) ||
           (?insdCDSEnd >= ?insdcStepEnd && ?insdCDSEnd <= ?insdcStepBegin) )
##  Then we look for a node close to the ones in the CDS in genome graph space (one step)
  ?node vg:linksForwardToForward ?nextNode .
  ?step2 a vg:Step ;
      vg:path/skos:closeMatch ?nextPath ;
        vg:node ?nextNode .
##  Where that node is on a uniprot matching sequence
  ?nextinsdCDS insdc:ctranslation ?nextSequence ;
        a insdc:Coding_Sequence ;
        faldo:location/faldo:begin/faldo:reference ?nextPath .
  ?uniprot up:sequence/rdf:value ?nextSequence .
  BIND(IF(?insdCDSBegin > ?insdcStepBegin, ?insdCDSBegin, ?insdcStepBegin - ?insdCDSBegin)/3 AS ?stepBeginInProteinSpace)
  BIND(IF(?insdCDSEnd > ?insdcStepEnd, ?insdcStepEnd, ?insdCDSBegin - ?insdcStepEnd)/3 AS ?stepEndInProteinSpace)
  ?uniprot up:annotation ?annotation .
  ?annotation a up:Active_Site_Annotation .
  ?annotation up:range ?annotationRegion .
  ?annotation rdfs:comment ?annotationText .
  ?annotationRegion faldo:begin/faldo:position ?annotationBegin .
  ?annotationRegion faldo:end/faldo:position ?annotationEnd .
  FILTER (?annotationBegin >= ?stepBeginInProteinSpace && ?annotationEnd < ?stepEndInProteinSpace )
}
```

Fig. 4. SPARQL query for retrieving nodes representing the coding nucleotides for an active site at protein level.

4 https://github.com/graph-genome/MatrixTubeMap/tree/splitsparql2

5 https://twitter.com/simonheumos/status/1169884828860239874

6 https://github.com/ekg/seqwish

```
<pg/zoom1> a vg:ZoomLevel ;
    vg:components <pg/zoom1/component1>,
        <pg/zoom1/component2>,
        <pg/zoom1/component3>,
        <pg/zoom1/component4> ;
    vg:zoomFactor 1 .
<pg/zoom1/component4> a vg:Component ;
    vg:bins <pg/zoom1/component4/bin12>,
        <pg/zoom1/component4/bin16>,
        <pg/zoom1/component4/bin17> ;
    vg:componentRank 4 ;
    vg:reverseComponentEdge <pg/zoom1/component3> .
<pg/zoom1/link1> a vg:Link ;
    vg:arrival <pg/zoom1/component2/bin1> ;
    vg:departure <pg/zoom1/component4/bin17> ;
    vg:linkPaths <6> .
<pg/zoom1/component2/bin1/cell1> a vg:Cell ;
    vg:cellRegion <6/region/1-1> ;
    vg:inversionPercent 0 ;
    vg:positionPercent 0.14 .
<pg/zoom1/component4/bin17> a vg:Bin ;
    vg:binRank 17 ;
    vg:cells <pg/zoom1/component4/bin17/cell28>,
        <pg/zoom1/component4/bin17/cell34>,
        <pg/zoom1/component4/bin17/cell40> ;
    vg:reverseBinEdge <pg/zoom1/component4/bin16> .
<6/region/1-1> a faldo:Region ;
    faldo:begin <6/position/1> ;
    faldo:end <6/position/1> .
```

Fig. 5. Example Turtle output of Component Segmentation..

3.3 Zero Extra Costs SPARQLable Genome Graphs - Spodgi

Converting and loading a genome graph into an RDF datastore can incur significant costs in storage. Spodgi[7] shows we can use python RDFLib to run SPARQL on native genome graph (odgi) data formats without extra storage related costs. If required, Spodgi can serialize the genome graph to Turtle[8], providing it on the Semantic Web in a triple store. For example a *SARS-CoV-2* pangenome graph, in an optimised native storage layer odgi will use 20mb of disk space, while the same converted to n-triples takes 7.2 GB. Even an optimized Turtle representation with the best xz compression is almost three times the size on disk.

4 DISCUSSION

vg RDF and pantograph RDF are pangenome ontologies based on Semantic Web technologies for describing the data structure in a consistent manner. The various use-cases using vg and pantograph RDF show that users can easily query pangenome across the border of data providers. SPARQL endpoint with data described in pangenome ontologies will encourage users to reuse queries in various tools that access genomic annotations on variation graphs.

ABBREVIATIONS

- **FAIR**: Findable, Accessible, Interoperable, Reusable
- **FALDO**: Feature Annotation Location Description Ontology
- **IRI**: Internationalized Resource Identifier
- **OWL**: Web Ontology Language
- **RDF**: Resource Description Framework
- **SPARQL**: SPARQL Protocol and RDF Query Language
- **Turtle**: Terse RDF Triple Language
- **UniProtKB**: Universal Protein Knowledgebase

ACKNOWLEDGEMENTS

We thank DBCLS for organizing the RDF summit and BioHackathons for fostering the initial discussions and developments. We also thank the participants of the COVID-19 online BioHackathon, who participated in a discussion on a pangenome browser and ontology. This work was supported in part by JSPS KAKENHI (grant number 19J21608). S.H. acknowledges funding from the Central Innovation Programme (ZIM) for SMEs of the Federal Ministry for Economic Affairs and Energy of Germany.

REFERENCES

Ballouz,S. et al. (2019) Is it time to change the reference genome? Genome Biol., 20, 159.

Beyer,W. et al. (2019) Sequence tube maps: making graph genomes intuitive to commuters. Bioinformatics, 35(24), 5318-5320.

Bolleman,J.T. et al. (2016) FALDO: A semantic standard for describing the location of nucleotide and protein feature annotation. J. Biomed. Semantics, 7, 1–12.

Eizenga,J.M. et al. (2020) Succinct dynamic variation graphs. bioRxiv,2020.04.23.056317, 1–6.

Garrison,E. et al. (2018) Variation graph toolkit improves read mapping by representing genetic variation in the reference. Nat. Biotechnol., 36, 875–879.

Kawashima,S. et al. (2018) NBDC RDF portal: a comprehensive repository for semantic data in life sciences. Database, 11, 1–11.

Lamprecht,A.-L. et al. (2019) Towards FAIR principles for research software. Data Sci., 1, 1–23.

Howe,K.L. et al. (2020) Ensembl Genomes 2020-enabling non-vertebrate genomic research. Nucleic Acids Res., 48, D689–D695.

[7] https://github.com/pangenome/spodgi
[8] https://www.w3.org/TR/turtle/

Sudmant,P.H. et al. (2015) An integrated map of structural variation in 2,504 human genomes. Nature, 526, 75–81.

The UniProt Consortium. (2019) UniProt: a worldwide hub of protein knowledge. Nucleic Acids Res., 47, D506–D515.

Wilkinson,M.D. et al. (2016) The FAIR Guiding Principles for scientific data management and stewardship. Sci. Data, 3, 160018.

Yokoyama,T.T. et al. (2019) MoMI-G: modular multi-scale integrated genome graph browser. BMC Bioinformatics, 20, 548.

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX uniprotkb: <http://purl.uniprot.org/uniprot/>
PREFIX uberon: <http://purl.obolibrary.org/obo/uo#>
PREFIX taxon: <http://purl.uniprot.org/taxonomy/>
PREFIX sp: <http://spinrdf.org/sp#>
PREFIX SLM: <https://swisslipids.org/rdf/>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX sio: <http://semanticscience.org/resource/>
PREFIX sh: <http://www.w3.org/ns/shacl#>
PREFIX schema: <http://schema.org/>
PREFIX rh: <http://rdf.rhea-db.org/>
PREFIX pubmed: <http://rdf.ncbi.nlm.nih.gov/pubmed/>
PREFIX patent: <http://data.epo.org/linked-data/def/patent/>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX orthodb: <http://purl.orthodb.org/>
PREFIX orth: <http://purl.org/net/orth#>
PREFIX obo: <http://purl.obolibrary.org/obo/>
PREFIX np: <http://nextprot.org/rdf#>
PREFIX nextprot: <http://nextprot.org/rdf/entry/>
PREFIX mnx: <https://rdf.metanetx.org/schema/>
PREFIX mnet: <https://rdf.metanetx.org/mnet/>
PREFIX mesh: <http://id.nlm.nih.gov/mesh/>
PREFIX lscr: <http://purl.org/lscr#>
PREFIX keywords: <http://purl.uniprot.org/keywords/>
PREFIX identifiers: <http://identifiers.org/>
PREFIX glyconnect: <https://purl.org/glyconnect/>
PREFIX glycan: <http://purl.jp/bio/12/glyco/glycan#>
PREFIX genex: <http://purl.org/genex#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX eunisSpecies: <http://eunis.eea.europa.eu/rdf/species-schema.rdf#>
PREFIX ensembltranscript: <http://rdf.ebi.ac.uk/resource/ensembl.transcript/>
PREFIX ensemblterms: <http://rdf.ebi.ac.uk/terms/ensembl/>
PREFIX ensemblprotein: <http://rdf.ebi.ac.uk/resource/ensembl.protein/>
PREFIX ensemblexon: <http://rdf.ebi.ac.uk/resource/ensembl.exon/>
PREFIX ensembl: <http://rdf.ebi.ac.uk/resource/ensembl/>
PREFIX ec: <http://purl.uniprot.org/enzyme/>
PREFIX dc: <http://purl.org/dc/terms/>
PREFIX cco: <http://rdf.ebi.ac.uk/terms/chembl#>
PREFIX chebihash: <http://purl.obolibrary.org/obo/chebi#>
PREFIX CHEBI: <http://purl.obolibrary.org/obo/CHEBI_>
PREFIX bibo: <http://purl.org/ontology/bibo/>
PREFIX allie: <http://allie.dbcls.jp/>
PREFIX GO: <http://purl.obolibrary.org/obo/GO_>
PREFIX orthodbGroup: <http://purl.orthodb.org/odbgroup/>
PREFIX vg: <http://biohackathon.org/resource/vg#>
PREFIX insdc: <http://ddbj.nig.ac.jp/ontologies/nucleotide/>
PREFIX faldo: <http://biohackathon.org/resource/faldo#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX up: <http://purl.uniprot.org/core/>
SELECT
  DISTINCT
   ?insdCDS #?insdCDSBegin ?insdCDSEnd ?step
            ?uniprot ?stepBeginInProteinSpace
  ?stepEndInProteinSpace ?annotationText
WHERE
{
#  Find CDS annotated by INDSC that do not match a
  UniProt protein.
 ?insdCDS insdc:translation ?sequence ;
          a insdc:Coding_Sequence ;
          faldo:location ?insdCDSLocation .
 MINUS {
   ?uniprotSequence rdf:value ?sequence .
 }

#  Get the range of this CDS and make sure the coordinates
  are on the
#  path we need later
 ?insdCDSLocation faldo:begin [ faldo:reference ?path ;
                                faldo:position ?insdCDSBegin] ;
                  faldo:end [ faldo:reference ?path ;
                                faldo:position ?insdCDSEnd] .

 ?step a vg:Step ;
    vg:path/skos:closeMatch ?path ;
   vg:node ?node ;
    faldo:begin [ faldo:reference/skos:closeMatch ?path ;
                                                  faldo:position
  ?insdcStepBegin ] ;
    faldo:end [ faldo:reference/skos:closeMatch ?path ;
                                faldo:position ?insdcStepEnd ] .
## I always forget how to interval ranges :(
  FILTER ( (?insdcStepBegin >= ?insdCDSBegin &&
  ?insdcStepBegin <= ?insdCDSEnd) ||
        (?insdCDSBegin >= ?insdcStepBegin &&
  ?insdCDSBegin <= ?insdcStepEnd) ||
                (?insdcStepEnd >= ?insdCDSEnd &&
  ?insdcStepEnd <= ?insdCDSBegin) ||
        (?insdCDSEnd >= ?insdcStepEnd &&
  ?insdCDSEnd <= ?insdcStepBegin) )
##  Then we look for a node close to the ones in the CDS in
  genome graph space (one step)
 ?node vg:linksForwardToForward ?nextNode .
 ?step2 a vg:Step ;
    vg:path/skos:closeMatch ?nextPath ;
  vg:node ?nextNode .
##  Where that node is on a uniprot matching sequence
 ?nextinsdCDS insdc:translation ?nextSequence ;
```

```
        a insdc:Coding_Sequence ;
        faldo:location/faldo:begin/faldo:reference
    ?nextPath .
  ?uniprot up:sequence/rdf:value ?nextSequence .
       BIND(IF(?insdCDSBegin > ?insdcStepBegin,
    ?insdCDSBegin, ?insdcStepBegin - ?insdCDSBegin)/3
    AS ?stepBeginInProteinSpace)
  BIND(IF(?insdCDSEnd > ?insdcStepEnd, ?insdcStepEnd,
    ?insdCDSBegin - ?insdcStepEnd)/3 AS
    ?stepEndInProteinSpace)
  ?uniprot up:annotation ?annotation .
  ?annotation a up:Active_Site_Annotation .
  ?annotation up:range ?annotationRegion .
  ?annotation rdfs:comment ?annotationText .
        ?annotationRegion faldo:begin/faldo:position
    ?annotationBegin .
         ?annotationRegion faldo:end/faldo:position
    ?annotationEnd .
   FILTER (?annotationBegin >= ?stepBeginInProteinSpace
    && ?annotationEnd < ?stepEndInProteinSpace )
}
```