The Genome Interval Algebra and its applications to the Sequence Ontology and Genome Databases

Mungall, C.J

July 20, 2009

Abstract

The advent of next generation sequencing technology heralds a transformation in healthcare and the life sciences. However, the proilferating number of datasets vary tremendously in how they encode valuable genomic information. Existing standards are loosely specified, largely underconstrained and do not provide efficient decision procedures for answering even basic genomic questions such as “how many introns are there in the genome of this organism?”. This means that programs written for one dataset frequently do not work with other datasets. This has been a major obstacle within bioinformatics, and the problem promises to get worse as datasets increase in size, number and richness.

In order to take full advantage of next generation genomics data, we need informatics methods to be based on agreed upon *formally specified* standards that can be implemented easily in a uniform fashion without ambiguity. These standards should be encoded as logical formulae, so that provably correct and efficient decision procedures can be used for query answering and validation.

In this paper we present the core of such a standard for sequence data. It consists of definitions of relations that hold between genomic intervals, and an alegbra for performing operations upon these intervals. We show how this algebra can be used to extend the Sequence Ontology (SO), generating a formal system that can be used by automated reasoning enginers both for validation of genomic datasets and for efficiently answering complex genomic queries over databases such as Chado and Ensembl.

1 Introduction

How many distinct introns are there in the human genome? What is the proportion of intergenic to genic regions across all sequenced eukaryotes? How many SNPs lie in coding regions vs in UTRs? How are single-exon genes distributed across phylogenetic lineages?

The union of publically available genome databases contain sufficient data to answer these questions, yet obtaining the answers is still a non-trivial task requiring significant bioinformatics expertise, despite the existence of friendly web front-ends allowing advanced data-mining queries such as EnsMart[1] and InterMine.

Why is this is a difficult problem? One reason is that genome databases consist of a minimum amount of explicitly *asserted* information, excluding information that can be deterministically *inferred*. For example, the existence of and genomic location of introns can be deduced from the existence of and location of exons; UTRs (Untranslated Regions) can be computed from the CDS (Coding Sequence) and the mRNA (messenger RNA); intergenic regions can be deduced from the position of genes; the quality of being "dicistronic" can be deduced from the cardinality of the relation between coding sequences and transcripts. Genome databases and their cognate exchange formats tend to be minimal in their explicit representations, which has the advantage of minimizing redundancy and space requirements, but this can be a hindrance to effective querying and programmatic use of genomic data.

Until now the deductive rules for inferring implicit features from asserted ones have until now never been encoded in a formal system, only in ad-hoc scripts and programming libraries. This is problematic, as the lack of a proof procedure means it is difficult to guarantee correctness of the resulting programs, and optimizations must be performed by hand. Additionally, this creates a resource bottleneck as these programs must be written by developers familiar with the relevant software libraries. These programs frequently break or give incorrect results in the presence of non-canonical central dogma gene models, or they may have built in assumptions that prevent their use across the major kingdoms of life. A case in point here is the *Drosophila* bi-strand trans-spliced gene $mod(mdg4)$[2]. Most databases and exchange formats fail to represent this faithfully in a consistent manner.

In this paper we propose a collection of precisely defined genomic relations, and an extension of the Sequence Ontology constructed using these relations, giving us a formal system for reasoning about genomic entities. This axiomatization can be used in database systems and ontological reasoning systems to answer questions about genomes, and to validate exchange formats and their data. We can also use the formal definitions to verify and unify programs and programming libraries that calculate certain topological and positional information about genomic features. Finally, the extended Sequence Ontology can serve as an entity in its own right, a crystalization of our current knowledge concerning the biological world at the genome sequence level.

The relations defined here are have their basis in two existing systems: The Region Connection Calculus (RCC-8) and the Allen Interval Algebra (AIA).

1.1 Region Connection Calculus (RCC-8)

The region connection calculus (RCC) is a system for qualitative spatial representation and reasoning. RCC abstractly describes regions (in Euclidian space, or in a topological space) by their possible relations to each other. RCC8 consists of 8 basic relations that can hold between two regions: *disconnected (DC), externally connected (EC), equal (EQ), partially overlapping (PO), tangential*

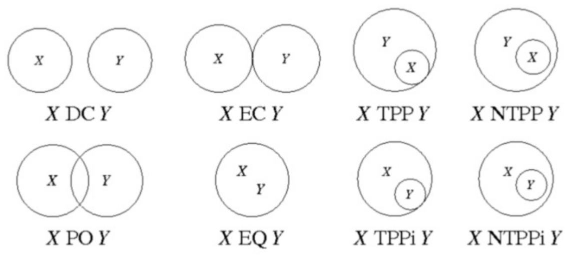


Figure 1: Relations in the Region Connection Calculus

proper part (TPP), tangential proper part inverse (TPPi), non-tangential proper part (NTPP) and *non-tangential proper part inverse (NTPPi).* See figure 1.

We can compose these relations using logical operators. For example, proper part (PP) is the union of TPP and NTPP.

RCC-8 is commonly used in geographical information systems. It could also be of use in reasoning about biological or biochemical entities.

1.2 Allen's Interval Algebra

Allen's Interval Algebra (AIA)[3] defines possible relations between time intervals that can be used as a basis for qualitative reasoning about temporal descriptions of events.

The AIA defines 13 pairwise disjoint base relations that capture all possible relations between two intervals. The 13 relations consist of 6 asymmetric relations: *precedes (p), meets (m), overlaps (o), finishes (f), contains (di), starts (s)*, their inverses: *precededBy (pi), metBy (mi), overlappedBy (oi), finishedBy (fi), containedBy*[1] *(c), startedBy (si)*, and the symmetric *equals (e)* relation.

Composing relations together gives a total possible 8192 relations. The composition operators are intersection (*cap*), union (*cup*) and complementation ($\neg$). For example, the union $p \cup pi$ holds whenever two time intervals have no time point in common.

Note that the Allen overlaps (o) relation does not correspond to what one might understand intuitively by "overlaps". The intuitive sense, that is, having a sub-interval in common is captured by $o \cup f \cup d \cup s \cup e \cup oi \cup fi \cup di \cup si$.

Whilst the AIA is intended for reasoning about temporal intervals, it is sufficiently abstract to be useful for reasoning about intervals in general, including genomic relations on a strand.

1.3 Sequence Ontology

The Sequence Ontology (SO)[4] was originally conceived of as a structured controlled vocabulary for genome databases and exchange formats. The SO consists

[1]sometimes called "during"

of broad terms and relationships for genome features types such as exon, gene, UTRas well as more specific types such as five_prime_coding_exon. Gradually the SO adoped more of an ontological flavour, for example, treating relations as logical axioms, adding disjointness axioms and employing logic-based definitions and a reasoner to compute subsumption relations between the terms.

At the core of SO is the is_a hierarchy, a collection of sentences such as mature_transcript is_a transcript, which, when treated as logical axioms, translates to x instance_of mature_transcript $\rightarrow$ x instance_of transcript, i.e., any instance of a mature transcript is necessarily an instance of a transcript. SO also declares other relations between terms, such as TSS part_of primary_transcript, indicating that all instances of TSSare necessarily part of some primary_transcript. The relations in SO are defined according to the methodology laid out in the OBO Relations Ontology[5].

SO includes lots of composite terms which have logical definitions[2], expressed in simple genus differentia form, i.e. *an X is a G that D*. For example, transposable_element_gene is defined as a gene that is part_of a transposable_element. The genus here is "gene" and the characteristic that differentiates this from other genes is being "part of a TE". Formally, this translates to a logically quantified axiom:

$$x \text{ instance\_of transposable\_element\_gene} \leftrightarrow x \text{ instance\_of gene} \wedge x \text{ part\_of transposable\_element}$$

These definitions are useful for both automatic classification within the ontology, and can be translated to database queries to automatically classify feature records. For example, even if a database does not explicitly assert a gene g to be a TE gene, it can be inferred to be one on the basis of its relations to a TE.

SO is currently lacking computable definitions for many terms such as:

- UTR
- region
- five_prime_UTR
- CDS
- intron
- five_prime_coding_exon

These definitions are specified in natural language, which means they cannot be used algorithmically. This is a major limitation, as it means we cannot query a database for all "intron" features, if these features have not been explicitly asserted.

SO has also historically used the part_of and member_of relations. These conform to standard mereological axioms, but are still somewhat underdefined. For example, the part_of relation which holds between TSSand primary_transcript- does this entail that the boundaries of the transcript enclude the TSS? In this case probably not, the parthood relation is more one of functional dependence; but in other cases parthood has a regional flavour.

[2]http://wiki.geneontology.org/index.php/SO:Composite_Terms

The SO is designed to be used in conjunction with sequence databases and exchange formats, such as Chado and GFF3.

1.4 Chado and GFF3

Chado is a relational model centered around the representation of genomic features[6], designed to be used in conjunction with SO. The Chado model allows for the representation of a collection of features, each of which is assigned a type from SO. Chado *reifies* types: i.e. they are part of the data, not the model.

In the core Chado genomic model, a chado dataset consists of the tuple $C = (F, G^F, G^L)$:

F is the set of n features:

$$F = \{f_n | n \in N\}$$

We also define an instantiation function $inst : F \to T$, where T is the set of types declared in SO, and a sequence function $seq : F \to S$.

The Feature graph set G^F defines an edge-labeled graph over the set of features

$$G^F \subseteq \{(f_s, f_o, r, rank) : f_s, f_o \in F, r \in R, rank \in \mathbb{N}_0\}$$

Where the set of edge labels R are also reified and declared in SO. The rank defines a total ordering of edges emanating from a feature.

The Location graph set G^L defines an interval position labeled graph between features and parent features:

$$G^L \subseteq \{(f_s, f_o, p) : f_s, f_o \in F, p \in P\}$$

where P is an interval range:

$$P \subseteq \{(min, max, rank, g, strand) : min, max \in I, strand \in \{+, -, 0\}\}$$

Chado models are instantiated as relational databases. There exists two exchange formats, Chado-XML and Chaos-XML that retain an isomorphic structure and identical semantics to the one specified above.

GFF3 is a tab-delimited exchange format for genomic data. It is informally specified, and can be considered a subset of Chado (if we ignore the seldom used split location option which has unclear semantics). Crucially, the feature graph in GFF3 is not edge-labeled. i.e.

$$G^{F2} \subseteq \{(f_s, f_o) : f_s, f_o \in F\}$$

The above formalism is under-constrained in that it admits many models of questionable validity, such as a datasets in which a single transcript has overlapping exons or where a TSS is 5' of the gene of which it is a part. As yet there is no formal semantics or decision procedure for deciding which subsets of C are valid. In addition there is no single unified mechanism for materializing implicit feature types such as introns and UTRs.

This means that programs frequently have to be adapted or rewritten to handle different datasets. This approach scales where the programmer to data ratio is high - this is acceptable for bioinformatics research, but not for implementing robust systems. Ideally we could use a formal specification of both the ontology and datasets together with generalized decision procedures to infer and validate in a robust manner.

1.5 Reasoning and Decision Procedures

Given a formal specification of some system, we can use automated decision procedures, known as *reasoning*, to infer new facts or to check for the validity of data with respect the specification.

For reasoning about the relations between temporal intervals, Allen's Interval Algebra provides a composition table. Given the relation $R1$ between x and y and the relation $R2$ between y and z, the composition table allows us to determine the relation between x and z. This can be applied recursively over a chain of relations. Together with a converse operation, this turns Allen's Interval Algebra into a relation algebra.

Satisfiability is NP-Complete with AIA[**?**], i.e. given a collection of intervals and the relations that hold between them we cannot in general compute if there are time values for which the relations are true. However, there are tractable sub-algebras for which efficient decision procedures exist[7].

RCC8 also employs a composition table[8].

A limited form of reasoning is currently used with the SO, implemented using the OBO-Edit[9] reasoner. The is_a hierarchy of composite terms is maintained automatically, and checks are made for violations of disjointness axioms. As yet the reasoner has not been applied to data.

In this paper we are primarily interested in extending the Sequence Ontology with new relations and reasoning over the results. There are 4 broad categories of use cases here, which we categorize according to the reasoning square (see table 1). We can use reasoners to determine new facts from basic axioms, or to determine the validity of a set of facts. We can perform these two tasks over ontologies, or over collections of facts derived from ontologies and datasets (such as GFF3 files).

Reasoners are frequently used with ontologies: for example, SO uses the OBO-Edit reasoner to automatically classify the ontology, and to find human errors in the ontology. As yet there have been limited uses of reasoners over genomic datasets, but possible applications include determining implicit features, and providing decision procedures to validate data exchanged using formats such as GFF3.

There are a number of reasoning systems that can be used with ontologies. We divide these into 3 broad categories: first-order logic (FOL) theorem provers,

	Entailment	**Validation**
Ontologies	Inference of relations in an ontology.	Checking for inconsistencies in an ontology
Data	Inference of implicit features. Examples: introns from exons, UTR from CDS and transcript	Validating databases and GFF3 files

Table 1: The reasoning square: Reasoners can be used to infer unstated knowledge (entailment) or to perform model checking (validation) to ensure there are no inconsistencies. These tasks can be performed on data or on ontologies

description logic reasoners and rule-based reasoners. Full first order logic is undecidable, so FOL theorem provers are used only in specific situations such as proving theorems within some general axiom system. The other approaches work by determining some subset of FOL, expressing the ontology (and possibly related data) using this subset and reasoning over the results.

1.6 Validation using SO

[TODO: move this to results?]

In theory Chado and GFF3 data can be validated using SO. In practice this has proved difficult to implement. Instead, "best practices" have evolved[10]. Whilst these are undoubtedly useful, formal validation is preferable than best practice, as this ensures automatic validation and no misiniterprations. It also scales better for wider communities of users, where best practices are more likely to be deviated from or misunderstood.

The reasons why formal validation using SO is difficult include:

1. **Open World Assumption**. In a defining contrast to data models, Ontologies make the open world assumption: they represent the patterns that exist in the world rather than those that exist in data. For example, SO might make the statement that all genes include at least 1 regulatory region. However, if we treat this as a cardinality constraint in a data model, then any dataset that lacks an associated regulatory region for a gene g will be treated as invalid. This means that one cannot simply translate an ontology to a data model. The open world model is fundamentally richer, in that it allows us to make inferences in the presence of unstated data: in our example we can infer that there is some regulatory region r associated with g in reality, even though it is not explicitly stated in the dataset. The open-closed world distinction often trips up programmers coming to ontologies from a data modeling perspective[11].

2. **All-Some Relations**. All relations in SO currently have all-some semantics. i.e. exon part_of transcript means that all exons are part of some transcript [formally: $\forall Exon(x) : \exists Transcript(t) \wedge x$ part_of t]. Note that this does *not* disallow a part_of link between an exon and some other

type of feature such as an intron. For this we would need an All-Only relation [formally: $\forall Exon(x) : x$ part_of $y \rightarrow Transcript(y)$]. Whilst it may be the case that some existing SO sentences should really have All-Only semantics, it would be over-interpreting to assume this for all of them. Thus SO would benefit from explicitly declaring All-Some vs All-Only semantics.

3. **Sequence Transformations**. A standard database gene model representation or its corresponding visualization is highly compressed in that there is a lot of unstated information. The genomic coordinates of a series of exons and a CDS implicitly represent a complex biological chain of events including transcription, splicing and translation. A formalization must take this additional contextual information into account.

4. **Lack of reasoning tools**. Most existing OWL reasoners are designed to work over an ontology in main-memory. This does not scale to genome databases.

[Something about existing approaches here. Mention object models and database schemas, e.g. Ensembl. Works for central dogma but becomes difficult for more complex gene models.]

1.7 Inferring implicit features using SO

In any genomic database or data exchange format many features are not explicitly stated. Or if they are, they may not be asserted using the most specific SO type possible. For example, the following SO types are rarely asserted directly:

- intron
- UTR
- 5'UTR
- exon of single exon gene
- polycistronic gene
- 5' coding exon

Introns are typically left implicit from exon positions. Exons are stated, but an exon is rarely stated explicitly to be an exon of a single exon gene.

Sometimes there is no consensus about what is implicit and explicit. For example, most databases manifest a CDS but leave start and stop codons as implicit. However, Ensembl reverses this pattern.

Type	Explicit in	Implicit in
exon	Ensembl, FlyBase, GenBank	
five_prime_coding_exon		Ensembl, FlyBase, GenBank

start_codon	Ensembl	FlyBase, Genbank
stop_codon	Ensembl	FlyBase, Genbank
CDS	Genbank, FlyBase[3]	Ensembl
intron	Genbank (sometimes)	Ensembl, FlyBase
intron	Genbank (sometimes)	Ensembl, FlyBase
UTR	Genbank (sometimes)	Ensembl, FlyBase
five_prime_UTR		Genbank, Ensembl, FlyBase

Table 2: Databases differ in which types are explicitly stated, and which are to be inferred programmatically.

At present some programs provides means to infer some of these features, but this currently patchy and incomplete. Also, they probably can't be used for efficient database queries.

This is related to validation too: for example, if a database provides both introns and exons, then the asserted intron locations could correspond to the locations of inferred exons (modulo closed world assumption).

Currently the logically transparent axioms in SO are insufficient to infer implicit features, or to validate data. As a step towards filling in this gap, we have defined precise relations for the genomic positioning and cardinality of relationships between features. This paper describes this set of relations, which constitute the Genome Interval Algebra (GIA).

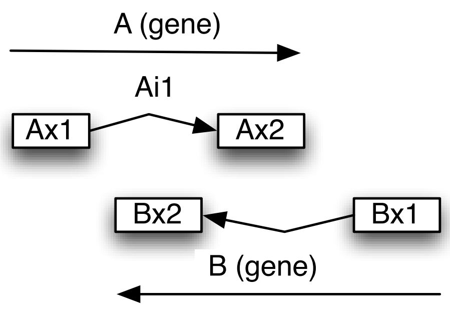
2 Results

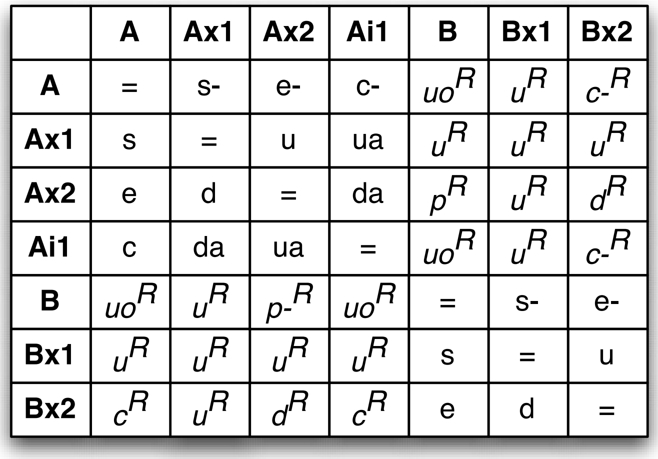
2.1 An Algebra of Genomic Intervals

We base the Genomic Interval Algebra (GIA) on the Allen Interval Algebra. RCC-8 does not take into account the intrinsic directionality/asymmetricality of genomic intervals. Whilst the Allen Interval Algebra is intended for temporal intervals, it can be used for spatial or genomic sequence intervals. We change some of the terminology of Allen (e.g. using "upstream" and "downstream" instead of "precedes" and "precededBy"), and take some terms from RCC-8 (e.g. "adjacent"), but use Allen-based definitions.

When considering the biological meaning of these relations, it is important to stress that they hold between *sequences* or *sequence intervals* and not the molecules that are the bearers of those sequences. For example, an RNA molecule or intron may exhibit connectedness/adjacency between bases at the secondary structure level. We consider these to be non-adjacent and disconnected at the sequence level. It may be desirable to eventually label these relations such that any ambiguity is removed, but we opt not to do this in the context of this paper.

[3]FlyBase actually uses polypeptide here





	A	Ax1	Ax2	Ai1	B	Bx1	Bx2
A	=	s-	e-	c-	uo^{R}	u^{R}	$c\text{-}^{R}$
Ax1	s	=	u	ua	u^{R}	u^{R}	u^{R}
Ax2	e	d	=	da	p^{R}	u^{R}	d^{R}
Ai1	c	da	ua	=	uo^{R}	u^{R}	$c\text{-}^{R}$
B	uo^{R}	u^{R}	$p\text{-}^{R}$	uo^{R}	=	s-	e-
Bx1	u^{R}	u^{R}	u^{R}	u^{R}	s	=	u
Bx2	c^{R}	u^{R}	d^{R}	c^{R}	e	d	=

Figure 2: Example of genomic sequence interval relations: two interleaved genes A and B on opposite strands. The lookup table shows the mnemonics for the relations between any two feature intervals. To determine the relation xRy, look up x as a row and y as a column, and read R from the cell. For example the first exon of A (Ax1) is upstream of the reverse-complement projection of all the exons of B.

In terms of the Basic Formal Ontology (BFO)[**?**], we consider sequences to be *generically dependent continuants* that *inhere* in multiple molecules. This means we can talk about "Jim Watson's Genome"[**?**] as individuals in the singular, even though there are trillions of copies of this genome in each of Watson's cells[4].

The core of the GIA consists of 16 basic relations that can hold between any two intervals on the same strand of a sequence, defined in terms of Allen relations. The Allen alebra treats intervals as primitives: we also provide alternate definitions in terms of junctions (the equivalent being time-points in a temporal calculus), yielding a junction calculus. We then extend the set of relations this to account for strandedness, deriving an additional 32 relations.

Figure 2 illustrates these relations with a simplified example.

2.1.1 Basic single-strand relations

We have declared 15 interval relations that can hold between two intervals on the same strand of a sequence, show in table 3.

Some relations are defined in terms of other relations using relation-intersection and relation-union operators. These have the normal semantics. i.e.

[4]Of course, a multi-cellular organism will have multiple genomes due to somatic mutations. We gloss over this here.

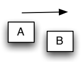
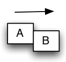
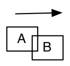
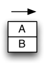
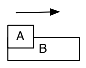
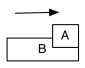
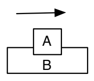
Relation	Allen Definition	Inverse
 u upstream_of		d [downstream_of]
a adjacent_to	$m \cup mi$	[symm]
ua upstream_adjacent_to 	m	da [downstream_adjacent_to]
uo upstream_overlaps 	o	do [downstream_overlaps]
o overlaps	$o \cup f \cup c \cup s \cup e \cup si \cup ci \cup fi \cup oi$	[symm]
$!o$ disconnected_from	$p \cup m \cup mi \cup pi$	[symm]
 $=$ coincident_with	e	[symm]
 s starts	s	$s-$ [started_by]
 f finishes	f	$f-$ [finished_by]
 c part_of	c	$c-$ [has_part]

Table 3: Core Relations in the Genome Interval Algebra. Glyphs depict the relation holding between A and B, with the strand indicated by an arrow. This table provides definitions based on Allen relations. We provide both biologist-friendly names (e.g. upstream of) and mnemonics (e.g. uo)

intersection: $x(R \cap S)y \leftrightarrow xRy \wedge xSy$
union: $x(R \cup S)y \leftrightarrow xRy \vee xSy$
complement: $x(!R)y \leftrightarrow \neg[xRy]$
Note that we do not take the same set of primitives as Allen; we choose our set based on utility within genome databases. This means that there is not an isomorphic correspondence between the GIA and Allen. We provide Allen-based definitions using relation intersection and and union. Unlike Allen, our set is not pairwise disjoint. For example, $ua = u \cap a$.

For the 15 core relations, the only meaningful comparisons are between points on the same strand on the same sequence (recall we treat the two strands of double stranded DNA as seperate sequences). If we want to compare two intervals on opposite strands we use relations defined via the RC function (see further on for a definition). This simplifies the algebra, but introduces additional relations for cross-strand comparisons.

Note the potential for a slight terminological problem: we use overlaps in a much broader sense than Allen. Our sense is compatible with the Relations Ontology definition[5], i.e. things overlap if they have some part in common (here the parts would be nucleotide bases). We also need to test for the cognitive adequacy of term upstream_of - the way we define it, it may be better called "completely upstream of", to make it clear that overlap is excluded. [TODO: we should discuss this. There are cases where it would be useful to have 3 relations: completely UO, partially UO and make UO the union of these two]

[authors note: this draft is still a bit inconsistent in whether it uses part of / has part or contains/contained by. The term part-of may be overloaded. For example, a TSS may be traditionally considered part of a transcript, even though it lies upstream of the transcript, rather than contained within it. Here part of seems to mean functional-part-of. Therefore it may be better if we used containment relations throughout as this is less ambiguous. We would probably make contains (hasPart) the basic relation and containedBy (partOf) the inverse, which inverts the current situation, so a few changes would have to be made to the tables]

The disconnected_from relation bears special mention. We denote this as $!o$ but it is not truly the negation of overlaps, because intervals on different sequences are treated as neither overlapping nor disconnected. [TBD : we can change this. It may make things simpler to treat these intervals as disconnected]

2.1.2 Junction-based definitions

We also provide definitions for all interval relations in terms of point-positions or junctions. A junction connects two bases (see definition below) - here we also consider the outermost ends of a region to be junctions [do we need a better term?].

We define two functions α and ω each of which maps an interval to a point (junction). The operators $<$ and $>$ are used to tell the ordering of pairs of junctions.

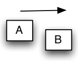
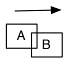
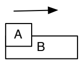
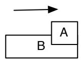
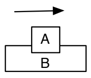
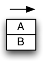
Relation	Junction Definition
 u upstream_of	$\omega(x) <= \alpha(y)$
ua upstream_adjacent_to	$\omega(x) = \alpha(y)$
a adjacent_to	$\alpha(x) = \omega(y) \vee \omega(x) = \alpha(y)$ (derived from: $ua \cup da$)
 uo upstream_overlaps	$\alpha(x) < \alpha(y) \wedge \omega(x) < \omega(y) \wedge \alpha(y) < \omega(x)$
o overlaps	$\omega(x) > \alpha(y) \wedge \alpha(x) < \omega(y)$
$!o$ disconnected_from	$\omega(x) <= \alpha(y) \vee \alpha(x) >= \omega(y)$ (derived from: $\neg o$)
 s starts	$\alpha(x) = \alpha(y) \wedge \omega(x) < \omega(y)$
 s finishes	$\alpha(x) > \alpha(y) \wedge \omega(x) = \omega(y)$
 c part_of	$\alpha(x) > \alpha(y) \wedge \omega(x) < \omega(y)$
 $=$ coincident_with	$\alpha(x) = \alpha(y) \wedge \omega(x) = \omega(y)$

Table 4: Junction-based definitions. Definitions are not shown for inverses, which can be trivially obtained by reversing x and y

For any interval, the start is before the end. Formally:

$$\forall x : Interval(x) \rightarrow \alpha(x) < \omega(x) \vee \alpha(x) = \omega(x)$$

(we allow for zero-length intervals here, i.e. junctions, see below)

Note that if $<$ and $>$ here do not operate on numbers. If we want a numerical interpretation then α and ω should map to a pair (seq,position), with $<$ and $>$ having normal semantics over position but always returning false when non-identical sequences.

Alternatively we could have made the functions map to integers, and write out the rules more explicitly; e.g. $\alpha(x) <= \omega(y) \wedge sameSeqAndStrand(x, y)$ for upstream_of . This is more akin to standard database representations.

We can translate junction-based definitions such as the following:

$$\omega(x) <= \alpha(y) \rightarrow x \text{ upstream\_of } y$$

Into composition rules involving primitive junction relations:

$$omega \cdot \text{ upstream\_of }^{J} \cdot alphaOf \rightarrow \text{ upstream\_of}$$

Here *alpha* and *omega* are a functional relations between a region and a junction, with inverses $alphaOf$ and $omegaOf$.

These additional relations give us an algebra which we call $GIA(J)$ [CHECK: no longer closed so is this still an algebra?]. Formalization in this fashion is useful in reasoning systems that allow for relation composition (e.g. OBO and OWL).

2.1.3 Derived Reverse-Complement Relations

The major difference between temporal and genomic intervals is that DNA is stranded. The GIA must therefore be an *extension* of the AIA to fully account for strandedness.

There are a number of ways to axiomatize strandedness. Our solution is to layer on to AIA relations rather than to generalize them. We treat a double-stranded DNA molecule as bearing *two* sequences, related by a reverse complementation operator. This is in contrast to traditional database representations, where a single DNA molecule is represented using a single sequence record, with reverse strand sequences extracted dynamically.

For any interval x on a DNA strand, there exists a "shadow" interval, $\mathsf{RC}(x)$ on the opposite strand. Some shadow intervals may not be biologically meaningful, e.g. for stranded features such as genes. However, for other features, e.g. repeats, either interval could be arbitrarily designated the shadow.

For any relation in the core set, $r \in R$, we can define a reverse-complement cognate, r^R. We obtain definitions for these automatically using the formula:

$$r^R(x, y) = r(x, \mathsf{RC}(y))$$

In table 5 we show only one RC relation, $\mathsf{upstream\_overlaps}^R$. This is identical to upstream_overlaps , but with RCapplied to the second argument. Note that unlike upstream_overlaps , this is a symmetric relation. Conversely, $\mathsf{part\_of}^R$ (not shown), **is** the inverse of $\mathsf{has\_part}^R$. See figure 2 for an illustration of why this is the case.

For any $r \in R$, we can further define a relation $r^U = r \cup r^R$. Table 5 illustrates this with $\mathsf{upstream\_overlaps}^R \cup \mathsf{upstream\_overlaps}$ which we call this $\mathsf{upstream\_overlaps}^U$. Arguably these derived relations actually represent the common use cases (e.g. Region-of-Interest queries in Genome Browsers[12][13]), so we should have intuitive names for them. However, from the point of view of axiomatisation, it is simpler to treat these as derived rather than basic relations.

Thus we have 15 relations in the core set (including inverse relations for non-symmetric relations). We declare relations for this core set, plus their RC equivalents, plus the union set. This gives us 45 relations in total. This may seem excessive but as we will see we will need most for our use cases.

2.1.4 Operations over sets of intervals

Neither RCC-8 not AIA dictate operations over *sets* of regions or intervals. We propose a simple extension in GIA for dealing with sets - for example, a set

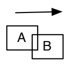
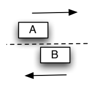
Relation	Definition	Inverse
uo upstream_overlaps 	$\alpha(x) < \alpha(y) \wedge \omega(x) < \omega(y) \wedge \alpha(y) < \omega(x)$	do [downstream_overlaps]
uo^R upstream_overlaps^R 	$\alpha(x) < \alpha(\mathsf{RC}(y)) \wedge \omega(x) < \omega(\mathsf{RC}(y)) \wedge \alpha(\mathsf{RC}(y)) < \omega(x)$	[symm]
uo^* upstream_overlaps^U	$uo^R \cup uo$	$uo^R \cup do$

Table 5: Example of Reverse Complementation cognate relations for the upstream_overlaps relation. Note the interaction between RC and inverse relations: in particular the inverse of uo^R does not correspond to a single named relation.

of exons in a transcript on the genome (not to be confused with the transcript itself, which is a single interval).

We use the same relations as for single intervals, but we must be careful how we define them. We can also define new relations that are only applicable to sets. For example, in figure 2 genes A and B stand in an o^R (overlaps) relation, even though the exons share no bases. However, we can introduce a new relation *interleaves*, defined such that the exons in A interleave (on the opposite strand) the exons in B. (note that it is important that we distinguish between the sets of exons interleaving and the genes overlapping).

For a relation R in GIC to hold between a set of intervals S and an interval I, it must hold between either a single element of S and I, or between all elements of S and I, depending on the relation. We also use a function *msr*, for the minimal spanning region for the set.

X overlaps Y: $\exists(x,y) : x \in X, y \in Y[x \text{ overlaps } y]$

X adjacent to Y: $\exists(x,y) : x \in X, y \in Y[x \text{ adjacent\_to } y] \wedge \neg X \text{ overlaps } Y$ (two subintervals must be adjacent, and none must overlap)

X interleaves Y:

$$\exists(x^1,x^2,y^1,y^2) : \quad x^1 \in X, x^2 \in X, y^1 \in Y, y^2 \in Y, \\ x^1 \text{ upstream\_of}^R\ y^2 \wedge x^1 \text{ downstream\_of}^R\ y^1 \wedge x^2 \text{ downstream\_of}^R\ y^2, \\ \neg X \text{ overlaps}^R\ Y$$

X upstream of Y: $msr(X)$ upstream_of $msr(Y)$

[These axioms have yet to be fully fleshed out].

2.1.5 Composition table

The full composition table for GIA(J) is too large to show. It includes

Transitivity of upstream of:

$$\textsf{upstream\_of} \cdot \textsf{upstream\_of} \rightarrow \textsf{upstream\_of}$$

starts and ends composes to containment:

$$\textsf{starts} \cdot \textsf{finishes} \rightarrow \textsf{part\_of}$$

The RC upstream of relation is transitive over downstream of (consider Ax1 u Bx2 d Bx1 in figure 2)

$$\textsf{upstream\_of}^R \cdot \textsf{downstream\_of} \rightarrow \textsf{upstream\_of}^R$$

[Include proofs from Prover9?]

2.2 Beyond the Interval Algebra: Sequences

Allen considers basic relations between pairs of intervals, and the combinations of those relations.

Here we introduce a relation is_consecutive_sequence_of , as a relation between sequence types. A sequence is a collection of adjacent non-overlapping sequences (thus sequences can be recursively composed).

$$\begin{aligned}
&T \ \textsf{is\_consecutive\_sequence\_of} \ U \leftrightarrow \\
&\quad \forall t : t \ \textsf{instance\_of} \ T \rightarrow [\quad \exists S_t : S_t = \{u_1, u_2, ..., u_n\} \ \wedge \\
&\quad\quad \forall u_i \in S_t : u_i \ \textsf{instance\_of} \ U \ \wedge \\
&\quad\quad \neg(\exists u_x, u_y \in S_t : u_x \ \textsf{overlaps} \ u_y) \ \wedge \\
&\quad\quad \forall u_x \in S_t[(u_x \ \textsf{starts} \ t \vee \exists u_{x-1} : u_{x-1} \ \textsf{adjacent\_to} \ u_x) \ \wedge \\
&\quad\quad (u_x \ \textsf{finishes} \ t \vee \exists u_{x+1} : u_{x+1} \ \textsf{adjacent\_to} \ u_x)] \\
&\quad]
\end{aligned}$$

Examples:

- $$\textsf{region is\_consecutive\_sequence\_of base}$$
- $$\textsf{CDS is\_consecutive\_sequence\_of codon}$$
- $$\textsf{mature\_transcript is\_consecutive\_sequence\_of exon}$$
- $$\textsf{polypeptide is\_consecutive\_sequence\_of amino\_acid}$$

Note that only spliced (mature) transcripts are necessarily made of consecutive sequences of exons (although some primary transcripts may be, i.e. intron-free ones).

There is a problem with the above relations and the current SO: we will get an inconsistency if additional reasonable axioms are added (such as making base and polypeptide disjoint). This is because regions in SO include polypeptides. Perhaps we can distinguish between na_regions and polypeptides? They seem fundamentally different. Or we can introduce a supertype of base and amino acid.

These axioms allow us to derive common programmatic operations from first principles: for example, the appending together of exon sequences to obtain a transcript sequence. Whilst this might seem trivial it is surprising how often programmers get this wrong, especially in the case of unusual gene models such as the bi-strand trans-spliced gene *mod*(*mdg*4)[2]. If programs were written according to a formal axiomatization (or derived from them) then mistakes like this could be avoided [save for discussion section]?

2.2.1 Reverse Complementation

We define the reverse complementation operator for a sequence recursively:

$$\begin{aligned}\mathsf{RC}(s) = r: \quad & [(x, s')\ \mathsf{splits}\ s \wedge x\ \mathsf{complement\_of}\ y \wedge (y, r')\ \mathsf{splits}\ r] \vee \\ & s\ \mathsf{complement\_of}\ r\end{aligned}$$

The definitions of splits is:

$$(h, t)\ \mathsf{splits}\ s \leftrightarrow h\ \mathsf{adjacent\_to}\ t \wedge h\ \mathsf{starts}\ s \wedge t\ \mathsf{finishes}\ s$$

Complementation is a symmetric relation between bases:

$$\begin{gathered} a\ \mathsf{complement\_of}\ t \\ t\ \mathsf{complement\_of}\ a \\ c\ \mathsf{complement\_of}\ g \\ g\ \mathsf{complement\_of}\ c \end{gathered}$$

The RC function is symmetric:

$$\mathsf{RC}(\mathsf{RC}(x)) = x$$

When applied to an interval, the RC function yields the cognate interval on the reverse complemented sequence.

2.2.2 Further derived relations

We can define a 3-ary relation between a, x and y such that a plays the role of connector in the adjacency of x and y:

$$connects(a, x, y) \leftrightarrow \textsf{upstream\_adjacent\_to}\ (a, x) \wedge \textsf{downstream\_adjacent\_to}\ (a, y)$$

Other 3-ary relations can be defined in this way

2.3 Transcription, Processing and Translation

So far we have been considering *genome intervals* only. However, we are also interested in inherited genomic features that are also present on gene sequences subsequent to the processes of transcription, splicing and translation. Within the context of a single gene expression event, the initiation of each of these processes happens sequentially, i.e.:

$$\alpha(transcription) < \alpha(splicing) < \alpha(translation)$$

(Here we are applying the start function α to temporal intervals)

Relations between features can change with each expression event. For example, in figure 2, the relation that holds between exon pairs changes from upstream_of to upstream_adjacent_to after splicing.

This means that we should always state the parent interval whenever stating a relation between features. For example, given a donor-acceptor splice site pair d_1, a_1 on a primary transcript t_1, we may write upstream_of (d_1, a_1, t_1), and for a corresponding processed transcript t_2 we would write adjacent_to (d_1, a_1, t_2). Table 6 shows relations for the previously shown example.

Relation	Subject	Object	Base
u	Ax1	Ax2	genome-1
u	Ax1	Ax2	primary_transcript-A
ua	Ax1	Ax2	mature_transcript-A
da	Ai1	Ax1	genome-1
da	Ai1	Ax1	primary_transcript-A
[no rel]	Ai1	Ax1	mature_transcript-A

Table 6: 3-ary relations for figure 2. Genome-1 is the name of the genome on which all the features are located. Note the final entry shows no relation between the intron and the spliced transcript [TBD: we may decide the intron is “present” as a junction, but this is unusual]

This treats each expression event as identity-preserving: the same exon is present through transcription and splicing, it is just the relations that change.

Alternatively, we can retain binary relations and treat each event as eliminating features and creating new ones (the so-called time slices or histories modeling paradigm[**?**])

[TODO: if splicing is identity preserving then we can't use the name of the transcript as the distinguishing factor in the 3rd argument. What is the least complicated way of resolving this?]]

For the axiomatization we use 3-ary relations.

2.3.1 Projection Function

We use a *projection function*[14] to map features onto a particular sequence.

$$proj : F \times F \rightarrow F$$

When we say $R(x, y, z)$ it means the same as $R(proj(x, z), proj(y, z))$. Sometimes for convenience we abbreviate this to $R(z(x), z(y))$.

This projection function will only map features that are a functional part of the sequence being projected onto. Thus when projecting exons onto a primary transcript, none of the exons will overlap on the primary transcript.

[TODO: def of projection]

2.3.2 Temporal sequence relations

- encodes - between a genome region and a primary transcript
- processed into - between a primary transcript and mature transcript

[this may be getting beyond the scope of interval relations, but is important for the broader context of reasoning with SO]

2.4 Extending the Sequence Ontology with genome interval relations

We can use the relations above to extend the SO. In particular, we can provide *logical definitions* for many core terms. These definitions can be used for both reasoning over the ontology as an artefact in itself, and to infer the presence of unstated genomic features.

2.4.1 Logical quantifiers

So far we have considered relations between individual intervals. SO represents feature types, so we need to consider relations at the type level. For example, if we say:

$$\mathsf{splice\_site\ adjacent\_to\ exon}$$

We must be clear about which of the following we mean:

1. all splice sites are adjacent to some exon ($\forall s : \mathsf{splice\_site}(s) \rightarrow \exists x : \mathsf{exon}(x) \wedge s\ \mathsf{adjacent\_to}\ x$)

2. all splice sites are adjacent to only exons ($\forall s, x : \mathsf{splice\_site}(s) \wedge s\ \mathsf{adjacent\_to}\ x \rightarrow \mathsf{exon}(x)$)
3. some splice sites are adjacent to an exon ($\exists s : \mathsf{splice\_site}(s) \rightarrow \exists x : \mathsf{exon}(x) \wedge s\ \mathsf{adjacent\_to}\ x$)
4. all exons are adjacent to some splice site ($\forall x : \mathsf{exon}(x) \rightarrow \exists s : \mathsf{splice\_site}(s) \wedge x\ \mathsf{adjacent\_to}\ s$)
5. all exons are adjacent to only splice sites ($\forall s, x : \mathsf{exon}(x) \wedge x\ \mathsf{adjacent\_to}\ s \rightarrow \mathsf{splice\_site}(x)$)

The first case (All-Some) is the most common. It is useful for reasoning over the ontology and inferring the presence of unstated features. The second (All-Only) is useful for detecting inconsistencies, particularly over datasets.

The middle case (Some-Some) is too weak to be useful and we ignore it. Note that Some-Some is perhaps the most similar to object-oriented programmiong semantics [do we need to get into OWA vs CWA here?].

The final two cases are reciprocals of the first two.

The Relations Ontology (RO)[15] introduced a methodology in which both type level relations are defined in terms of instance level relations. Note that if we introduce type level counterparts to all instance level genome relations defined thus far, we will have 60 * 2 = 120 relations. For simplicity, in this document we will use the instance level relation names and either indicate the quantifier explicitly, or assume an All-Some quantifier.

The RO also introduced temporal qualification at the type level. For example, if $R(X, Y)$, then it is the case that $\forall x : x\ \mathsf{instance\_of}\ X \rightarrow \exists y, t : x\ \mathsf{part\_of}\ y@t \wedge y\ \mathsf{instance\_of}\ Y$.

As noted previously, we must take into account the changes that happen in time through transcription and splicing. However, rather than quantifying explicitly over time, we will qualify each type-level relation with the sequence type on which that relation holds.

Thus for example it is not enough to state an all-some relation:

$$\mathsf{splice\_site\ overlaps\ intron}$$

Because this is not true for splice sites on processed transcripts. We will qualify such sentences as follows:

$$\mathsf{splice\_site\ overlaps\ intron}@^{\mathsf{primary\_transcript}}$$

Which means:

$$\begin{aligned} \forall x : x\ \mathsf{instance\_of\ splice\_site} \rightarrow \exists y, z : \quad & proj(x, z)\ \mathsf{overlaps}\ proj(y, z) \wedge \\ & y\ \mathsf{instance\_of\ splice\_site} \wedge \\ & z\ \mathsf{instance\_of\ primary\_transcript} \end{aligned}$$

2.4.2 Reciprocal links

We want to say things like "all primary transcripts are encoded by some gene" AND "all genes encode some primary transcript".

There are 3 different approaches here, wherever a reciprocal link is appropriate:

(i) create two axioms, one in each direction. E.g. $\forall t : PT(t) \rightarrow \exists g[G(g) \wedge enc(g,t)]$ and $\forall g : G(g) \rightarrow \exists t[PT(t) \wedge enc(g,t)]$

(ii) use axiom qualifier to declare links as holding true in inverse direction (link qualifiers in OE, semi-deprecated).

(iii) make this a special property of the encodes type-level relation.

We will aim for (iii) but implement (i) for now. This makes the graph more verbose (and could potentially causes issues for simplistic tools that use the SO). Users who don't care about the details of reciprocal quantification and just want a rough ideal how things are connected may find this confusing.

2.4.3 Logical Definitions

Logical definitions provide necessary and sufficient conditions in computable form. We have created nnn new definitions based on the GIA relations. Table 7 shows a subset of these

Type	Genus	Differentia
five_prime_coding_exon	coding_exon	overlaps start_codon
five_prime_UTR	UTR	upstream_adjacent_to CDS
five_prime_intron	intron	overlaps five_prime_UTR
UTR_intron	intron	overlaps UTR
twintron	intron	part_of intron
start_codon	codon	starts CDS
noncoding_exon	exon	disconnected_from CDS
interior_exon	exon	adjacent_to five_prime_splice_site $\wedge$ adjacent_to three_prime_splice_site $\wedge$ disconnected_from splice_site

Table 7: Examples of logical definitions of SO terms using GIC relations. Quantifiers are taken to be All-Some unless otherwise noted. The semantics of a genus differentia definition (T,G,D) are such that $T(x) \leftrightarrow G(x) \wedge D(x)$

Because the definitions are both necessary and sufficient, they can be used to construct queries on genome databases. For example, to find 5' coding exons, make a conjunctive query for coding exons that overlap start codons. overlaps can be translated to a genome coordinate query. The query may need further expansion of start codon or coding exon is not materialized in the database.

2.4.4 Maximal Regions

It may be tempting to define a UTR as

$$\textsf{part\_of mRNA} \wedge \textsf{ disconnected\_from CDS}$$

However, this actually defines what SO calls a UTR_region - a portion of UTR.

We need to state that this is the maximal portion of mRNA region that is bounded by CDS

As a next attempt we might try:

$$\textsf{part\_of mRNA} \wedge \textsf{ adjacent\_to CDS} \wedge (\textsf{ starts mRNA} \vee \textsf{ finishes mRNA})$$

(or, equivalently $UTR = 5'UTR \vee 3'UTR$)

However, this fails to account for internal UTRs

Similarly, we may be tempted to define the coding region of an exon as

$$\textsf{overlaps } CDS \wedge \textsf{ overlaps exon}$$

However, this type definition includes overlapping *portions* of coding_region_of_exons.

The current solution is to use a 3-ary relation[5], maximally_overlaps , such that:

$$a \textsf{ maximally\_overlaps } x, y \leftrightarrow \forall b : [b \textsf{ overlaps } a \rightarrow b \textsf{ overlaps } x \wedge b \textsf{ overlaps } y]$$

We can also suuply a position-based definition:

$$\begin{aligned} a \textsf{ maximally\_overlaps } (x, y) \leftrightarrow & & (1)\\ & \alpha(a) = max(\alpha(x), \alpha(y)) \wedge & (2)\\ & \omega(a) = min(\omega(x), \omega(y)) & (3)\end{aligned}$$

maximally_overlaps is functional in that applying it to two regions only yields a single maximal overlap region.

We can also define a subtraction operator, which could potentially yield sets of intervals

[5]Although OWL cannot handle 3-ary relations, this can be automatically expanded out to a more complex OWL expression, see supplementary

2.4.5 Translation

We introduce a translation relation, that holds both between a CDS and a polypeptide, but also between individual codons and amino acids.

This allows us to define codons as mRNA regions that overlap 3 bases and translate to a single codon:

$$\mathsf{codon} = \mathsf{region} \wedge \ \mathsf{part\_of}\ (some)\mathsf{mRNA} \wedge \ \mathsf{overlaps}\ (3)base \wedge translatesTo(1)\mathsf{amino\_acid}$$

2.4.6 Cardinality Constraints

We define an exon of a single exon gene as an exon that is indirectly encoded by a single exon gene. A single exon gene is defined as a gene that indirectly encodes 1 exon.

A polycistronic mRNA is a mRNAthat $\mathsf{has\_part}$ n CDS, where $n >= 2$. We say it has minimum cardinality 2.

2.4.7 Attributes

SO includes various *attribute terms* that can be applied to sequences.

Some of these are actually attributes of the molecule. Others are attributes that could be defined using the relations defined above: for example, *overlapping*.

In OBO our definition of *overlapping* looks like this:

```
[Term]
id: SO:0000068
name: overlapping
def: "An attribute describing a gene that has a sequence that overlaps the sequence of anoth
is_a: SO:0000067 ! gene_to_gene_feature
intersection_of: SO:0000400 ! sequence_attribute
intersection_of: quality_of SO:0000704^overlaps(SO:0000704) ! attribute_of (gene that overla
```

i.e.

$$qualityOf some(gene\text{that } \mathsf{overlaps}\ somegene)$$

It may be simpler to define *overlapping gene* as a subtype of gene, rather than defining the attribute of overlappingness.

Note that overlaps has to be defined as non-reflexive to avoid having everything be overlapping to itself. This seems easier than introducing an awkwardly named “proper-overlaps” non-reflexive counterpart. We do depart from normal mereological usage here (I think.. [CHECK]).

2.4.8 Regions and Junctions

SO distinguishes between regions and junctions. These pairwise disjoint types exhaustively partition sequence features (i.e. all features are one or the other, and nothing is neither). [we state this in the ontology using both disjoint from and union of constructs]

We define a region:

$$\mathsf{region} = \mathsf{feature} \wedge \ \mathsf{is\_consecutive\_sequence\_of}\ (\mathsf{base})$$

And state additional necessary conditions:

$$\mathsf{region}\ \mathsf{adjacent\_to}\ (2)\mathsf{junction}$$

(note that this may need to be relaxed for circular sequences)

We do not define junction but state necessary conditions only:

$$\mathsf{junction}\ \mathsf{adjacent\_to}\ (2)\mathsf{base}$$

Here we use cardinality constraints: every junction is adjacent to two bases (here, the outermost points of a region are neither junctions nor features).

Note we could define junction using the functions introduced earlier:

$$x\ \mathsf{instance\_of}\ \mathsf{junction} \leftrightarrow \alpha(x) = \omega(x)$$

[A couple of issues here: we defined α, ω and adjacent_to as applying to proper intervals only. Should we extend to junctions? This has a few implications for relations. We have to be careful how we define relations to give intuitive results. E.g. two identical junctions would be coincident, but not overlapping]

2.4.9 A requirement for logical rules in SO

Presently in the extended SO we have axioms such as TSS upstream_of primary_transcript. Given the above we may want to say TSS upstream_of $proj$(primary_transcript, genome). Here we abbreviate this as TSS upstream_of genome(primary_transcript).

However, this is in fact an extremely weak statement - it is stating that every TSS is upstream of *some* primary transcript, without stating any constraint about the relationship between TSS and transcript. In fact we want to say that a TSS lies upstream of the same transcript that it regulates.

If we treat the existing part of relation in SO as meaning "functional part of" (i.e. x fpo y : every x has a function fx that is required for y to fulfil its function), then what we want to say is:

$$\forall x \mathsf{TSS}(x) \rightarrow \exists y : \mathsf{transcript}(y) \wedge fpo(x,y) \wedge \ \mathsf{upstream\_of}\ (x, \mathsf{genome}(y))$$

And

$$fpo(x,y) \wedge \mathsf{TSS}(x) \wedge \mathsf{transcript}(y) \rightarrow \ \mathsf{upstream\_of}\ (x, \mathsf{genome}(y))$$

This latter rule is of the sort that is extremely practical for checking validiting of data: given a GFF file we can test for badly specified TSS features that are not upstream of the transcript they are part of.

2.5 Reasoning over the GIA and SO

Even if we consider just the core 16 basic GIA relations, we see that this does not correspond to any of the tractable Sub-algebras of OWL (for example, the adjacent_to relation, defined in terms of Allen as $m \cup mi$ appears in none of the sub-algebras). It therefore follows that the extensions such as the one for reverse complementation would fall in the tractable subset.

This means the satisfiability problem is NP-complete for the GIA: given any sufficiently large collection of intervals and GIA relations between them, we cannot tell if there exists a collection of interval number positions that satisfy these relations.

This may on the surface appear problematic, but in fact the satisfiability problem does not have real-world correlate in the post-genomic world where genomic intervals are specified in numeric coordinates rather than qualitative interval relations. Given numbers, satisfiability is trivial. The satisfiability problem over qualitative relations was more relevant in the days of physical mapping[16].

There are more relevant use cases for the GIA when we use it to extend the SO. These use cases can be divided according to the reasoning square:

1. Finding all entailed relations in an ontology
2. Checking validity within an ontology
3. Querying a dataset using SO and GIA relations
4. Checking validity of a dataset

The first two tasks operate entirely over ontologies. Ontologies specify what is general about intervals and thus do not contain particular numeric ranges. Databases (here the term is used generically for any collection of genomic information) in contrast typically do specify numeric ranges, but as we shall see the qualitative GIA relations are still useful.

All tasks make use of a full axiomatization of SO extended with GIA relations and definitions. This axiomatization falls within the boundaries of first order logic plus numbers. This means there is no tractable decision procedure for testing validity of sentences. We are therefore interested in finding subsets of FOL for which there are efficient decision procedures, and seeing how much of the extended SO we can axiomatize within this subset.

There are various reasoners that implement decision procedures for the OWL-2 language. This gives us many of the constructs we need for the extended SO. However, notable absences include:

1. relation intersection and union

2. relation definition in terms of numeric ranges
3. rules
4. n-ary relations

This means we cannot encode all junction-based definitions. For some, such as upstream_of , we can model this using role chains over junction-based definitions (see previous section). However, we cannot encode all relations like this without relation intersection and/or union.

2.5.1 Finding all entailed relations in an ontology

Given an ontology representing a set of types T and a set of relations R, we want to determine the set of sentences S entailed by the set of sentences asserted in the ontology:

$S \subseteq \{(s, r, o)| \in R, s, o \in T\}$

Where R is the set of type-level relations.

Reasoners which implement efficient decision procedures for this problem are useful (but not essential) for ontology maintenance, particular where composed classes are concerned. This is also useful for databases - the pre-computed set S can be loaded into a database to enhance queries (see below).

We can ask *How are gene and polypeptide related?* and get the answer that all polypeptides indirectly_encoded_by some gene (but not the inverse: not all genes encode a pp), see figure 3.

[note: this is a nice example but it doesn't illustrate GIA relations so it should probably be moved to other SO documentation. A better example may be deriving 5'UTR ua startCodon based on 5'UTR ua CDS s- startCodon and ua.s- $\rightarrow$ ua. TODO!!]

This is because we have a chain: polypeptide translated_from CDS part_of $mRNA$ is_a $matureTranscript$ pro

And the relation axioms:

1. indirectly_encoded_by $\leftarrow$ processed_from $\cdot$ encoded_by (holds-over-chain rule)
2. indirectly_encoded_by $\leftarrow$ is_a $\cdot$ indirectly_encoded_by (propagation of all-some relations over isa)
3. indirectly_encoded_by $\leftarrow$ part_of $\cdot$ indirectly_encoded_by (propagation over part of)
4. indirectly_encoded_by $\leftarrow$ translated_from $\cdot$ indirectly_encoded_by (propagation over translated from)

(apply the compositions from right to left on the chain). See figure 3.

Also: five_prime_UTR upstream_adjacent_to CDSfrom the relation intersection axion

$$\text{upstream\_adjacent\_to} \leftarrow \text{adjacent\_to} \wedge \text{upstream\_of}$$

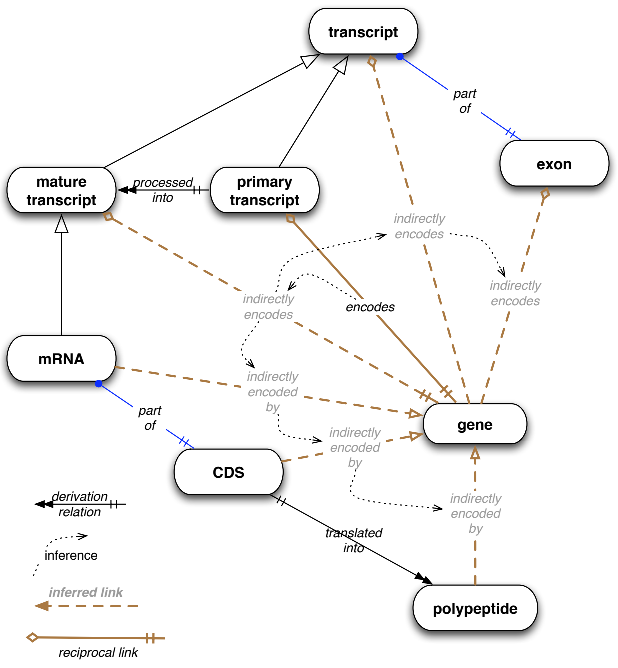


Figure 3: Type level reasoning: new type-level relationships are inferred using relation composition axioms in SO

We can use either a rule-based reasoner or a description logic reasoner for inferring non-redundant unstated relations. For the DL reasoning we used the standard translation of the SO to OWL-DL and used a variety of DL reasoners. We also used the OBO-Edit rule-based reasoner (which is currently used by the SO Editors).

The OWL reasoners were able to infer subsumption relations where cardinality is involved - e.g. *dicistronic transcript* is_a *polycistronic transcript*; the OE reasoner needs extended to do this.

Both reasoners were unable to make inferences where n-ary relations are involved.

2.5.2 Checking the validity of an ontology

OWL-DL is more suited than OBO as a language for checking the validity of models. However, our interval based definitions are still incomplete.

At this time we could find no inconsistencies between axioms within the extended SO. This is not surprising as the SO is carefully scrutinized by its editors prior to each release, and automated procedures are currently in use with the existing axioms. We still expect that the extended SO presents more precise axioms on which these checkers can operate.

TODO: strawman deliberately broken examples

2.5.3 Making inferences over and checking validity of data

[TODO: move section on validation from introduction here?]

Reasoners are commonly to make inferences over and check the validity of ontologies. A more relevant use case for bioinformatics is to perform the same operations over databases (sets of data), using the ontology as a specification. This is more of a challenge as the sizes of datasets are several orders of magnitude greater than the ontologies themselves. The datasets are certainly too large for generalized FOL theorem provers, so we must instead focus on tractable subsets. We explore two such subsets: an OWL-DL subset and a rule-based subset that can be used to extend relational database queries.

For the OWL-DL subset, we implemented a converter that takes GFF3, Ensembl, Chado etc and writes OWL ABox axioms. We also use the OWL-DL conversion of SO. As OWL cannot handle n-ary relations, we translate n-ary relations to binary ones by introducing additional instances to model the relation. Relation intersection and union are turned into weaker subProperty axioms.

Note that we cannot specify arbitrary rules in OWL - these are required to state for example that a TSS is always upstream of the genome projection of the transcript it regulates. We can extend OWL with SWRL rules to encode this. In OBO we can use a relation intersection, or rules.

See supplementary section for details of the OWL-DL conversion.

For certain rules, such as the one that states that the 5'UTR must be upstream and adjacent to the CDS in the same processed transcript, we can use an extension of DLs called *Description Graphs.* DGs are new, and we have not yet explored this fully.

One of the main limitations of OWL-DL for our purposes is the lack of ability to provide arithmetic-based definitions for relations. All genome database specify intervals numerically. Whilst it is possible to avoid numbers in OWL-DL by explicitly declaring all bases and *connected to* relations between them this is not very practical for genomes with billions of base pairs. Even then the lack of relation intersection and union constructs limits us from using OWL to answer even the most basic interval-based queries. [**?**]

TODO: more experiments on GFF3 expressed as OWL

2.5.4 Rule-based approaches

There are efficient decision procedures for the Horn-rule subset of first-order logic. The most efficient decision procedures are implemented in relational database systems. Whilst the relational model is relatively inexpressive in comparison to OWL-DL, it is by no means a strict subset. There are many things that are difficult or impossible in description logics that are trivial in the relational model, such as defining relations in terms of intersection and union operators, and providing definitions in terms of numbers.

These features make the relational model attractive for the GIA. For any relational genome database that makes use of some predicate or predicates for positioning of genomic features, we can generate relational queries for each of the GIA relations. Query answering is highly efficient, especially when the relational db is optimized, for example using Nested Containment Lists[17].

The major limitation of relational systems is the lack of mutually recursive predicate definitions. For this we must use extensions of the RM, such as Datalog (which extends the RM with mutually recursive predicate definitions) and Prolog (which extends Datalog with other features).

However, it is possible to use a hybrid approach to get something approximating the best of both worlds. We can avoid some of the need for recursive querying by precomputing recursive predicates in advance using an extension of the RM. For example, we can use the OBO-Edit reasoner to pre-compute all inferred relations in SO, and use the resulting lookup table in standard relational queries. Alternatively we can expand recursive predicates at query time, although this introduces a downstream dependency and the results may be less efficient:

For example, the following rule specifies that the TSS of a transcript must be upstream of that transcript:

$$fpo(x, y) \wedge \mathsf{TSS}(x) \wedge \mathsf{transcript}(y) \rightarrow \ \mathsf{upstream\_of}\ (x, \mathsf{genome}(y))$$

This can be automatically translated to Chado SQL which can be used to implement a database constraint:

Listing 1: Chado SQL Constraint: this query returns a list of all invalid TSSs

```
SELECT
 *
FROM
 TSS,
 transcript,
 fr.part_of
WHERE
 TSS.feature_id=part_of.subject_id AND
 transcript.feature_id=part_of.object_id AND
 NOT EXISTS (SELECT *
         FROM feature_upstream_of AS uo
         WHERE uo.subject_id=TSS.feature_id AND uo.object_id=transcript.feature_id);
```

Here the TSSand transcriptpredicates are translated to sub-queries over the SO lookup table (called cvtermpath in Chado). We also need to translate α and ω in our formalism to the native predicates used for locations in the database schema.

We can also infer implicit features such as:

- single-exon genes
- polycistronic genes

This means we can translate arbitrary complex queries involving SO terms that are not explicitly asserted in the database. For example: find all exons nested in introns on genes on the opposite strand upstream of the foo gene.

(TABLE HERE SHOWING COUNTS OF VARIOUS SO TYPES IN ENSCORE HUMAN AND FLYBASE)

See listing 2 in supplementary section for details

The hybrid approach is only useful in certain circumstances: we cannot precompute a lookup table for all features in a database.

Note that the RM does not require installation and maintenance of a relational database system: there are a number of lighweight in-memory databases that can be used.

3 Discussion

3.1 Making SO more rigorous

In describing the GIA and extending SO we came across portions of SO that were in need of more precise definitions. E.g.

[get this from tracker]

- intergenic_region- genes inside genes? strands? Bits between gene and origin or end of sequence
- splice_site- This was ambiguously defined, it was not clear whether the splice site was a region or a junction. It has been clarified to be a region, and a new term splice junction added

yadda

3.2 Circular genomes

At this time the calculus does not properly accounts for the circular genomes. It will need to be extended and generalized.

E.g. definition of region. Change it so that it is not necessarily border by two junctions. Introduce jointly exhaustive pairwise disjoint subtypes non-circular region and circular region.

3.3 Comparison with Allen

The Allen Algebra is qualitative. The relations we have defined have both quantitive/numeric definitions as well as qualitative definitions. This reflects the different uses for which we have for the relations. For type-level reasoning within an ontology we stick with qualitative relations. Typically when we have genome data we have numeric positions information, so the quantitive definitions can be used.

Continuous vs discrete. Nucleotides are discrete.

Reverse complementation.

4 Conclusion

We have demonstrated that the GIA relations defined here are of use in providing more rigorous computationally transparent definitions in the SO. The composition table is useful for making inferences over the ontology. However, for making inferences over data it is simpler to use number-based definitions rather than a composition table. Similar conclusions have been made about GIS databases and RCC8[**?**].

We found the OWL-DL language useful for reasoning over an ontology, but of limited use in making inferences and validating datasets. This is likely to be the most relevant scenario for the majority of genomics applications. In contrast, the relational model provides a surprisingly high degree of expressivity, especially when combined with logic programming approaches to precompute lookup tables.

Using our approach we have demonstrated how the SO can be used to make complex queries over either Chado and Ensembl relational databases, and also to validate the data contained within these databases. Our approach could easily be extended to other genomics database schemas. We have also demonstrated a validation and inference tool that does not require a relational database implementation, and can work off of GFF3 datasets.

5 Methods

We used OBO-Edit[9] to extract subsets of and make extensions to the Sequence Ontology. Some axioms were hand-edited on the obo file.

We used the OBO-Edit reasoner to determine the link path between types in SO.

We used the blipkit to extract feature data from GFF3/Chado/Ensembl and to write to other languages, including OWL-DL.

CGL[14]

5.1 Availability

The extension is called working draft.obo. Common Logic, Prover9 and OWL versions can be automatically generated.

Test sets

6 Acknowledgements

7 Supplementary Material

7.1 Translation to OWL-DL

7.1.1 Translation of Ontology

We use the standard obo to owl translation. We are using some obo 1.3 extensions here, which are transated as follows

The maximal overlap operator is translated to universal quantification over the inverse of overlaps (which is itself) such that $\mathsf{coding\_exon} = \mathsf{region} \wedge \ \mathsf{maximally\_overlaps}\ (\mathsf{CDS}, \mathsf{exon})$ is translated to:

```
CodingExon subClassOf (overlaps ONLY (overlaps SOME CDS AND overlaps SOME Exon))
```

We do not translate any definitions involving $<$, $>$, α or ω, restricting ourselves to qualitative reasoning. TODO: check current status of reasoning over interval domains in OWL

7.1.2 Translation of Feature Graphs (instance data)

The translation for a Chado feature graph is as follows:

1. Each f, t pair in FI is treated as an IndividualOf axiom
2. Each $f_i f_j r$ triple in G^F is treated as a fact (i.e. triple)
3. Each $f_i f_j pos$ triple in G^L is treated using the n-ary relations pattern, creating an Individual for the location, with α and ω as FunctionalProperties. We also try a simple representation in which the universe of discourse consists of only a single sequence and strand. Here α and ω are attached directly to the feature individuals.

this is really a whole paper in itself..

7.2 Converting SO definitions to Relational Queries

Given a mapping of core SO types to a relational schema, we can automatically translate some SO definitions to relational queries or views.

We can also arbitrarily compose these together and combine them with other constraints. For example, we can specify a query for single exon genes that are inside introns of genes on the opposite strand on chromosome 17:

$$\mathsf{single\_exon\_gene} \wedge \ \mathsf{overlaps}^{R}\ \mathsf{intron} \wedge \ \mathsf{overlaps}^{U}\ hschr17$$

with this automatically translated to a Chado or Ensembl SQL query, which can be issued over the network.

7.2.1 Ensembl auto-derived query for single_exon_gene

SO says:

$$\mathsf{single\_exon\_gene} = gene \wedge \ \mathsf{indirectly\_encodes}\ (1)\mathsf{exon}$$

The SO definition uses a cardinality=1 constraint, and cardinality=1 constraints are translated to relational queries of the form $R(X,Y) \wedge \neg(R(X,Y2) \wedge \neg(Y2 = Y))$, which gets expanded to SQL as follows. Note that the SO definition also uses indirectly encodes, which here gets translated to a gene-exon relationship.

Listing 2: Ensembl SQL

```
SELECT
 gene_stable_id_1.stable_id
 FROM
 exon_transcript exon_transcript_1 ,
 transcript transcript_1 ,
 transcript_stable_id transcript_stable_id_1 ,
 gene gene_1 ,
 gene_stable_id gene_stable_id_1
 WHERE
 transcript_1.transcript_id = exon_transcript_1.transcript_id AND
 transcript_stable_id_1.transcript_id = exon_transcript_1.transcript_id AND
 gene_1.gene_id = transcript_1.gene_id AND
 gene_stable_id_1.gene_id = transcript_1.gene_id AND
 NOT EXISTS (
             SELECT
              *
             FROM
              exon_transcript exon_transcript_2 ,
              transcript transcript_2 ,
              transcript_stable_id transcript_stable_id_2 ,
              gene gene_2 ,
              gene_stable_id gene_stable_id_2
             WHERE
              transcript_2.transcript_id = exon_transcript_2.transcript_id AND
              transcript_stable_id_2.transcript_id = exon_transcript_2.transcript_id AND
              gene_2.gene_id = transcript_2.gene_id AND
              gene_stable_id_2.gene_id = transcript_2.gene_id AND
              gene_stable_id_2.stable_id = gene_stable_id_1.stable_id AND
              exon_transcript_2.exon_id != exon_transcript_1.exon_id);
```

7.2.2 Chado auto-derived definition of 3' UTR intron

TODO

8 References

References

[1] A. Kasprzyk, D. Keefe, D. Smedley, D. London, W. Spooner, C. Melsopp, M. Hammond, P. Rocca-Serra, T. Cox, and E. Birney. Ensmart: a generic system for fast and flexible access to biological data. *Genome Res*, 14(1): 160–9, 2004. 1088-9051 Journal Article.

[2] Sima Misra, Madeline A Crosby, Christopher J Mungall, Beverley B Matthews, Kathryn S Campbell, Pavel Hradecky, Yanmei Huang, Joshua S Kaminker, Gillian H Millburn, Simon E Prochnik, Christopher D Smith, Jonathan L Tupy, Eleanor J Whitfied, Leyla Bayraktaroglu, Benjamin P Berman, Brian R Bettencourt, Susan E Celniker, Aubrey D N J de Grey, Rachel A Drysdale, Nomi L Harris, John Richter, Susan Russo, Andrew J Schroeder, Sheng Qiang Shu, Mark Stapleton, Chihiro Yamada, Michael Ashburner, William M Gelbart, Gerald M Rubin, and Suzanna E Lewis. Annotation of the Drosophila melanogaster euchromatic genome: a systematic review. *Genome Biol*, 3(12):RESEARCH0083, 2002.

[3] J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26:832–843, 1983.

[4] K. Eilbeck, S. E. Lewis, C. J. Mungall, M. D. Yandell, L. D. Stein, R. Durbin, and M. Ashburner. The sequence ontology: a tool for the unification of genome annotations. *Genome Biology*, 6(5), 2005.

[5] Barry Smith and Anand Kumar. Controlled vocabularies in bioinformatics: a case study in the gene ontology. *Drug Discovery Today: BIOSILICO*, 2 (6):246–252, 2004. TY - JOUR.

[6] Christopher J. Mungall, David B. Emmert, and The FlyBase Consortium. A chado case study: an ontology-based modular schema for representing genome-associated biological information. *Bioinformatics*, 23(13):i337–346, 2007. doi: 10.1093/bioinformatics/btm189. URL `http://bioinformatics.oxfordjournals.org/cgi/content/abstract/23/13/i337`.

[7] Andrei Krokhin, Peter Jeavons, and Peter Jonsson. Reasoning about temporal relations: The tractable subalgebras of allens interval algebra. *Journal of the ACM*, 50:2003, 2003.

[8] A G Cohn, B Bennett, J Gooday, and N Gotts. Representing and reasoning with qualitative spatial relations about regions. In O Stock, editor, *Temporal and spatial reasoning*. Kluwer, 1997.

[9] John Day-Richter, Midori A Harris, Melissa Haendel, Gene Ontology OBO-Edit Working Group, and Suzanna Lewis. Obo-edit–an ontology editor for biologists. *Bioinformatics*, 23(16):2198–2200, Aug 2007. doi: 10.1093/bioinformatics/btm112. URL `http://dx.doi.org/10.1093/bioinformatics/btm112`.

[10] Karen Eilbeck and Suzanna E. Lewis. Sequence ontology annotation guide. *Comparative and Functional Genomics*, 5:642–647, 2004. URL `http://www.hindawi.com/GetArticle.aspx?doi=10.1002/cfg.446&e=cta`.

[11] A. R. Ruttenberg, J. R. Rees, and J. S. Luciano. Experience using owl dl for the exchange of biological pathway information in: workshop 2005 owl: Experiences and directions. *Galway, Ireland*, 2005.

[12] L. D. Stein, C. Mungall, S. Shu, M. Caudy, M. Mangone, A. Day, E. Nickerson, J. E. Stajich, T. W. Harris, A. Arva, and S. Lewis. The generic genome browser: a building block for a model organism system database. *Genome Res*, 12(10):1599–610, 2002. 1088-9051 Journal Article.

[13] S. E. Lewis, S. M. Searle, N. Harris, M. Gibson, V. Lyer, J. Richter, C. Wiel, L. Bayraktaroglir, E. Birney, M. A. Crosby, J. S. Kaminker, B. B. Matthews, S. E. Prochnik, C. D. Smithy, J. L. Tupy, G. M. Rubin, S. Misra, C. J. Mungall, and M. E. Clamp. Apollo: a sequence annotation editor. *Genome Biol*, 3(12):RESEARCH0082, 2002. 1465-6914 Journal Article Review Review, Tutorial.

[14] Mark Yandell, Chris J Mungall, Chris Smith, Simon Prochnik, Joshua Kaminker, George Hartzell, Suzanna Lewis, and Gerald M Rubin. Large-scale trends in the evolution of gene structures within 11 animal genomes. *PLoS Comput Biol*, 2(3):e15, Mar 2006. doi: 10.1371/journal.pcbi.0020015. URL `http://dx.doi.org/10.1371/journal.pcbi.0020015`.

[15] B. Smith, W. Ceusters, J. Kohler, A. Kumar, J. Lomax, C.J. Mungall, F. Neuhaus, A. Rector, and C. Rosse. Relations in biomedical ontologies. *Genome Biology*, 6(5), 2005. URL `http://genomebiology.com/2005/6/5/R46`.

[16] Richard M. Karp. Mapping the genome: some combinatorial problems arising in molecular biology. In *STOC '93: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 278–285, New York, NY, USA, 1993. ACM. ISBN 0-89791-591-7. doi: http://doi.acm.org/10.1145/167088.167170.

[17] Alexander V Alekseyenko and Christopher J Lee. Nested containment list (NCList): a new algorithm for accelerating interval query of genome alignment and interval databases. *Bioinformatics*, page btl647, 2007. doi: 10.1093/bioinformatics/btl647. URL `http://bioinformatics.oxfordjournals.org/cgi/content/abstract/btl647v1`.