# DAG-Edit & OBO

- Come on a magical journey of discovery with me, and learn all the new things DAG-Edit can do now that we have the OBO format...

# The Wonderful World of Relationship Types

- OBO has imbued relationship types with all kinds of new and amazing powers.

# Range

- Relationship types now have a “range” attribute.
  - The range is a term. Only terms that are subclasses of the range term can be the target of a term relationship type.

# Range

- So, let's say we have the relationship type *has_sequence*. The range of *has_sequence* is sequence. That means that a relationship with the type *has_sequence* can only have parents that are subclasses of the term *sequence* (like *rna_sequence*, *dna_sequence*, etc).

Legal:


Illegal:

# Range

- This gives us a lot more ability to make sure the ontologies are consistent. This becomes really important when you're dealing with an ontology with a lot of relationship types (like the Sequence Ontology, or an anatomy, or the Phenotype Ontology, or any ontology with “cross products”).

# Domain

- Relationship types also can specify a domain.
  - The domain is a term. Only terms that are subclasses of the domain term can be the source of a term relationship type.

# Domain

- So, let's say we have the relationship type *has_color*. The domain of *has_color* is *visible_object*. That means that a relationship with the type *has_color* can only have children that are subclasses of the term *visible_object* (like *chicken*, *rock*, *leonardo_da_vinci*, etc).

Legal:


Illegal:

# Is Cyclic

- It is possible to say whether or not a relationship type can be used to create a cycle. It is illegal to create a cycle of relationships out of a relationship type that is not marked “is cyclic”.
- However, it is still legal to create cycles using multiple acyclic relationship types in combination.

# Is Transitive

- Transitive relationship types are relationship types such that:
  - If A has relationship X to B, and B has relationship X to C, by definition, A has relationship X to C.
  - is_a and part_of are transitive, which is why the true path rule works.

# Is Symmetric

- Symmetric relationship types are relationship types such that:
  - If A has relationship X to B, B has relationship X to A.
  - Precise identity relationships (like mathematical equality) are symmetric (if A=B, B=A). is_a and part_of are not symmetric.

# Namespaces

- It is now possible to assign terms (and term relationships) to a “namespace”.
  - Other things, like categories and dbxrefs, will be assignable to namespaces someday.

# Namespaces

- A namespace can be thought of in two ways
  - A namespace refers to the logical ontology to which a term belongs, regardless of where the ontology is stored. Namespaces let us know which ontology obsolete terms belong to, since we don't have any parentage info.
  - A namespace refers to the file in which a term (or relationship, or whatever) should be stored. The OBO save mechanism lets you save your session into several files based on namespaces.

# Namespaces

- Why bother with namespaces instead of assigning file names?
  - Terms might have been loaded from a source that doesn't have a proper name, like a database, so we use a more abstract designation
  - Terms in different files might still belong to the same namespace (if, for example, you created a sub-ontology in a special file, and wanted to merge it with another ontology).

# Relationship Namespaces

- A relationship can be marked with a different namespace than it's parent or child.
  - This can be used to create a distinct file for relationship types that don't belong in an existing ontology. For example, you could a bunch of links between two ontologies, you could keep those links in a third file by giving them a different namespace than the other two ontologies.
  - Relationships with no namespace are assigned to the namespace of the child term.

# Inverse Necessity

- It is possible to mark a relationship as necessarily true.
  - A necessarily true relationship *must always* be true. (ie *human_finger* is necessarily part_of *human_hand*, but *shoe* is not necessarily part_of *casual_outfit*, because a shoe is sometimes part_of a casual outfit, but someone might go barefoot).

# Inverse Necessity

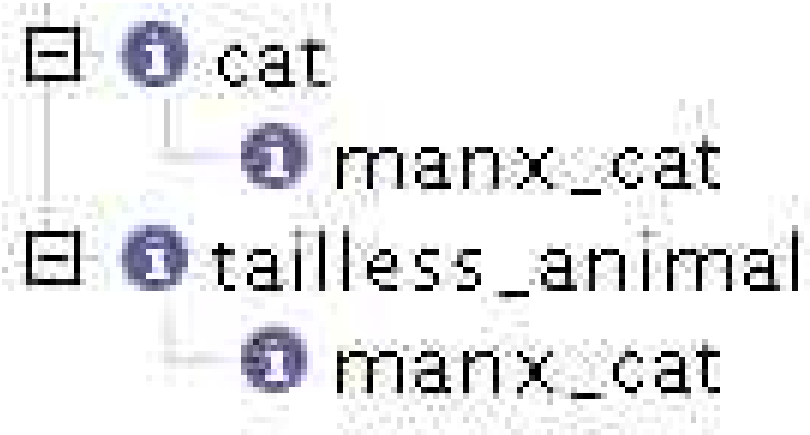
- It is possible to mark a the inverse of a relationship as necessarily true. A relationship may be inverse necessarily true with being necessarily true
  - For example: The relationship ***wheel*** **part_of** ***car*** is inverse necessarily true. Something with no wheels is not a car, but a wheel can still be a wheel without being part of a car.

# Completeness

- A “complete” term specification is one where anything that matches the term specification is *by definition* an instance of that term.

# Example: An Incomplete Spec

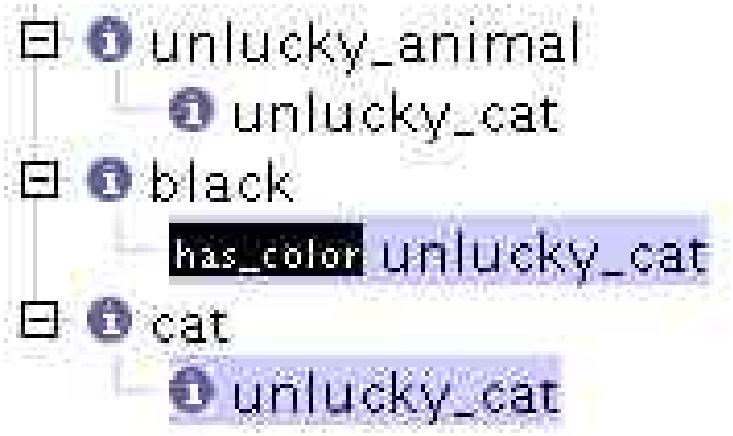
- The following term specification of manx_cat is incomplete:




- Because being a tailless_animal and being a cat does not automatically make something a manx_cat (it might be cat_with_severed_tail).

# Example: An Complete Spec

- The following term specification of unlucky_cat is complete:




- The relationships **has_color** ***black*** and **is_a** ***cat*** are enough to constitute a complete definition, because anything that is black and a cat is an unlucky cat.
- A complete term spec can contain relationships that don't contribute to the completeness (like **is_a** ***unlucky_animal***).

# The Parent Plugin

- There's a single easy place to set all these relationship attributes: The Parent Plugin.

# Fun with Obsolete Terms

- It is now possible to create a link between an obsolete term and the terms that should be used to replace it.

# New Built-In Relationship Types

- disjoint_from
- inverse_of

# disjoint_from

- A is *disjoint_from* B if no instance of A can ever be an instance of B.
  - *smart_guy* is disjoint_from *dumb_guy*
  - *visible_object* is disjoint_from *abstract_concept*
  - disjoint_from applies to subclasses of the specified terms. So if *george_w* is_a *dumb_guy*, *smart_guy* is disjoint_from *george_w*

# inverse_of

- inverse_of applies only to relationship types. It's easier to explain with examples:
  - *part_of* is the inverse_of *has_part*
  - *is_a* is the inverse_of *has_subclass*
  - *has_color* is the inverse_of *is_color_of*
- inverse_of is symmetrical. If A is the inverse of B, B is the inverse_of A.

# Conclusion: How does this help us

- All of these OBO-centric concepts are already present in the GO, they just aren't explicitly encoded. Hence the ongoing discussions about whether part_of means necessarily part of; sometimes it does, sometimes it doesn't, we just had no way of making it clear).
- With this additional information, we can now map to OWL and interact with reasoners much more effectively.