Foundations of Knowledge Representation in Cyc

- Why use logic?
- CycL Syntax
- *Collections and Individuals* (#$isa and #$genls)
- Microtheories

This section covers two concepts fundamental to the Cyc ontology: collections and individuals. We’ll also cover two fundamental predicates related to these concepts: #$isa and #$genls.
A collection is a type of thing, a kind of thing, or a class of things. Things which belong to a collection are called its *instances*.

Each collection is characterized by some feature or features that all of its instances share. For example the collection of all persons is a collection of all instances that share the properties that make something a person. Cher, Mario Andretti, Bill Clinton, and Abraham Lincoln are all instances of #$Person.

Some other collections include: the collection of all towers, the collection of all space stations and the collection of all movie directors.
Individuals

- An individual is a single thing, *not* a collection.
- Individuals do *not* have instances.
- Individuals may have *parts*.
- Some individuals:
  - #$EiffelTower
  - #$Mir
  - #$OrsonWelles
  - #$UnitedStatesMarineCorps

In contrast, an individual is a single thing, not a collection. #$Cher is an individual, and so is the #$EiffelTower.

Individuals do not have instances, but they may have parts. If you take a piece of the Eiffel Tower you don’t have an instance of the Eiffel Tower, you have a part of it. That’s one way to tell the difference between an individual and a collection.

Other individuals include: the space station Mir, Orson Wells, and the United States Marine Corps. That last one might surprise you. Why is the United States Marine Corps an individual and not a collection? Let’s look at that more closely.
Consider Joe the marine. Joe is a member of the Marine Corps, but he’s not an instance. Joe is a *part* of the Marine Corps. It would not make sense to say that Joe is an *instance* of the United States Marine Corps; he is not a Marine Corps himself. The United States Marine Corps is an individual, specific organization. There is only one United States Marine Corps. It has parts (for example, Joe) but not instances.

In contrast, we also have the collection #$UnitedStatesMarine. That collection is comprised of all human members of the United States Marine Corps. It has instances, each of which is an individual marine, like Joe.
Remember: collections can have instances, but not parts. Individuals can have parts but not instances. For example, the collection of all towers can have instances, each of which is a tower, but the Eiffel Tower can have parts.
Everything is an instance of some collection.

Every collection is, at minimum, an instance of #$Collection. So for example, the collection of all towers is an instance of #$Collection. The collection of all military personnel is an instance of #$Collection.

Every individual is, at minimum, an instance of #$Individual. So for example, the #$EiffelTower is an instance of #$Individual. #$OrsonWells is an instance of #$Individual.
Collections of Collections
and Collections of Individuals

• Some collections whose instances are individuals:
  – #$Tower
  – #$Person
  – #$Dog

• Some collections whose instances are collections:
  – #$ArtifactType
  – #$Collection

• Some collections with instances of both types:
  – #$ProprietaryConstant
  – #$DocumentationConstant

Now it gets more complicated: collections can have either individuals or collections as their instances.

For example, the collection of all towers has individual towers as its instances. The collection of all dogs has individual dogs as its instances.

But the collection #$ArtifactType has collections as its instances. For example, the collection of all computer programs is an instance of #$ArtifactType. The collection of all pieces of pottery is an instance of #$ArtifactType. So the instances of artifact type are themselves collections.

There are some special collections with instances of both types. Primarily these are collections of concepts themselves, where those concepts may indicate either collections or individuals.
Disjoint Collections

- Collections which have no instances in common are disjoint.

The collection of all dogs and the collection of all cats have no members in common. Having no instances in common is the definition of disjointness. Therefore, the collection of all dogs and the collection of all cats are disjoint. We express this in CycL using the predicate #$disjointWith, as in the formula along the bottom of the slide.
To express that something is an instance of a collection in CycL, we use the predicate #$isa. A formula of the form (#$isa X Y) means that X is an instance of collection Y. For example #$EiffelTower is an instance of the collection of all towers, #$Canada is an instance of the collection of all countries, #$Cher is an instance of the collection of all persons, and #$UnitedStatesMarineCorps is an instance of the collection of all modern military organizations.
• $(#$genls X Y)$ means
  “Every instance of collection X is also an instance of collection Y.”

  – $(#$genls #$Dog #$Mammal)
  – $(#$genls #$Tower #$FixedStructure)
  – $(#$genls #$ModernMilitaryOrganization #$Organization)

• Sometimes expressed in Cyclish® as:
  “Y is a genls (generalization) of X.”
  “X is a spec (specialization) of Y.”

To express that one collection is subsumed by another, we use the CycL
constant #$genls. A formula of the form $(#$genls X Y)$ means that every
instance of the first collection, X, is also an instance of the second collection,
Y. In other words, Y subsumes X; Y is a generalization of X.

For example, every instance of #$Dog is also an instance of #$Mammal;
#$Mammal is a generalization of #$Dog.

Sometimes this is expressed in Cyclish by simply saying Y “is a genls of” X,
or X “is a spec” (or specialization) of Y. So we might say, as in the second
example, that #$FixedStructure is a genls of #$Tower. Or we might say
#$Tower is a spec of #$FixedStructure. In either case we mean the same
thing: every instance of #$Tower is also an instance of #$FixedStructure.
The #$genls relationship is transitive.

So, since #$Dog has as a generalization, #$Mammal (meaning that every instance of the collection of all dogs is also an instance of the collection of all mammals), and #$Mammal is a specialization of #$Animal, we can conclude that #$Dog is a specialization of #$Animal – that is, every #$Dog is also an instance of #$Animal.

Similarly, since #$Computer (the collection of all computers) is a specialization of the collection #$ComputationalSystem, and #$ComputationalSystem is a specialization of #$PhysicalDevice, we can conclude that every instance of #$Computer is also an instance of #$PhysicalDevice. That’s what we mean when we say that #$genls is transitive.
Here’s a sample piece of the #$genls hierarchy.

 #$Lighthouse is a specialization of #$Tower, which itself is a specialization of #$FixedStructure, which in turn is a specialization of #$Individual.

 Because #$genls is transitive, that means that #$Lighthouse is also a specialization of #$Individual. And if we know about any particular lighthouse, we can conclude that it is also an instance of #$Tower, of #$FixedStructure and of #$Individual.
#$isa$, on the other hand, is not transitive.

Consider the number five. It’s an instance of #$PositiveInteger$. #$PositiveInteger$, in turn, is an instance of #$InfiniteSetOrCollection$. But five is not an instance of #$InfiniteSetOrCollection$.

Let’s take another example. #$Cher$ is an instance of #$Person$. #$Person$ is an instance of #$Collection$, but #$Cher$ is not an instance of #$Collection$.

So #$isa$ is not transitive.
Remember . . .

- Because every instance of a collection is also an instance of the collection’s genls, the following statements are true:
  - “#$isa$ transfers through #$genls.”
  - “#$isa$ does NOT transfer through #$isa.”

Because #$genls$ is transitive, every instance of a collection is also an instance of any #$genls$ (or generalization) of that collection. By the definition of #$genls$, it is the case that #$isa$ transfers through #$genls$, but #$isa$ does not transfer through #$isa$. An example follows.
What can we conclude about #$Rover the dog?

If we know only that #$Rover is an individual dog (an instance of #$Dog), what else can we conclude about #$Rover given this #$genls hierarchy?

The relationships in dotted lines here represent instancehood or #$isa relationships, and the solid orange lines represent #$genls or generalization relationships.

Since #$isa transfers through #$genls, if we know that #$Rover is an instance of #$Dog, we can conclude that #$Rover is also an instance of #$Mammal, since #$Dog is a specialization of #$Mammal. And so we can go up the chain. We can conclude that #$Rover is an instance of #$Animal and an instance of #$Individual and an instance of #$Thing.

Notice that we cannot conclude that #$Rover is an instance of #$BiologicalSpecies because #$isa does not transfer through #$isa. #$Dog is an instance of #$BiologicalSpecies, in contrast to being a specialization of #$Mammal.
This kind of inheritance is very powerful. It allows us to conclude many things from the single assertion that #$Rover is an instance of #$Dog, because of #$Dog’s place in the #$genls hierarchy. Here's a more complete list of collections of which #$Rover is (through inheritance) an instance.
Consider the formula (#$genls #$Dog #$Dog), the collection of all dogs is a generalization of the collection of all dogs. Is this true? It just means that every instance of #$Dog is an instance of #$Dog. That’s true, and will be true not just for #$Dog, but for any other collection. That means that #$genls is reflexive.
Is #$isa reflexive?

Consider

(#$isa #$Dog #$Dog) NOT reflexive

However, consider

(#$isa #$Collection #$Collection) Not anti-reflexive, either

#$isa, however, is not reflexive. Consider: the collection of all dogs is not an instance of the collection of all dogs -- that collection is not itself a dog. However, there are examples where formulas of this form are true. The collection of all collections is itself a collection. Therefore #$isa is not anti-reflexive.
In summary, we’ve looked at collections and individuals and how to distinguish them. We’ve looked at the concepts of #$isa and #$genls in CycL. We discovered that #$genls is transitive and reflexive, but that #$isa isn’t transitive or reflexive. These concepts are fundamental to knowledge representation in CycL.