

When and Why to use a Classifier?

Alan Rector

with acknowledgement to

Jeremy Rogers, Pieter Zanstra, & the GALEN Consortium

Nick Drummond, Matthew Horridge, Hai Wang in CO-ODE/HyOntUSE

Information Management Group Dept of Computer Science, U Manchester

Holger Knublauch, Ray Fergerson, ... and the Protégé-Owl Team

rector@cs.man.ac.uk

co-ode-admin@cs.man.ac.uk

www.co-ode.org

protege.stanford.org

www.opengalen.org

When to use a classifier

1. At author time: As a compiler

- Ontologies will be *delivered* as “pre-coordinated” ontologies to be used without a reasoner
- To make extensions and additions quick, easy, and responsive, distributed developments, empower users to make changes
- Part of an *ontology life cycle*

2. At delivery time: As a service:

- Many fixed ontologies are too big and too small
 - Too big to find things; too small to contain what you need
 - Create them on the fly
- Part of an *ontology service*

3. At application time: as a reasoner

- Decision support, query optimisation, schema integration, ..., ..., ...
- Part of a *reasoning service*

When to use a classifier 1: Pre-coordinated delivery classifier as compiler

- **The life cycle**

- Gather requirements, sketch, experiment
- Establish patterns – design a “language”
 - Criteria for success: What a subject domain expert can learn in a few days



- Bulk authoring

- Classification

- Quality assurance



Development & evolution



- Commit classifier results to a pre-coordinated ontology & deliver

- Polyhierarchies (Protégé, DAG-Edit, OWL-Lite, RDF(S), Topic Maps, ...
 - Query and use with you favourite tool

Commit Results to a Pre-Coordinated Ontology

The screenshot displays the Protege ontology editor interface. At the top, two panels show the 'Subclass Relationship' for 'Asserted Hierarchy' and 'Inferred Hierarchy'. Both panels list the following classes: owl:Thing, Domain_entity, Probe_Categories_for_demo_and_test, ValuePartition, and ValueSelector. Below these panels is a table with two columns: 'Class' and 'Changed superclasses'. The table lists several classes and their associated changes:

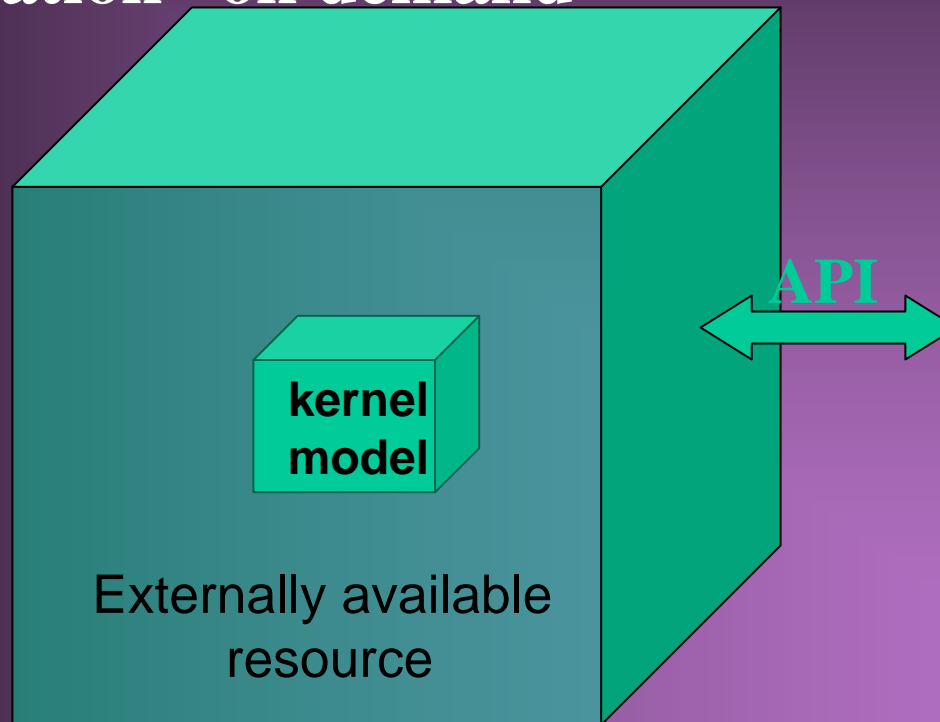
Class	Changed superclasses
Aortic_stenosis	Added Disorder_of_FMA_heart
Aortic_valve	Added Structural_parts_of_heart
Cardiac_chamber	Added Structural_parts_of_heart
Disorder_of_FMA_heart	Moved from Disorder, Probe_Categories_for_demo_a...
Half_heart	Added Structural_parts_of_heart
Pericarditis	Added Disorder_of_Clinical_heart
Pericardium	Added Clinical_parts_of_heart

An orange arrow points from the text 'Assert ("Commit") changes inferred by classifier' to the 'Assert selected change(s)' dialog box, which is currently open over the table.

Assert
("Commit")
changes
inferred by
classifier

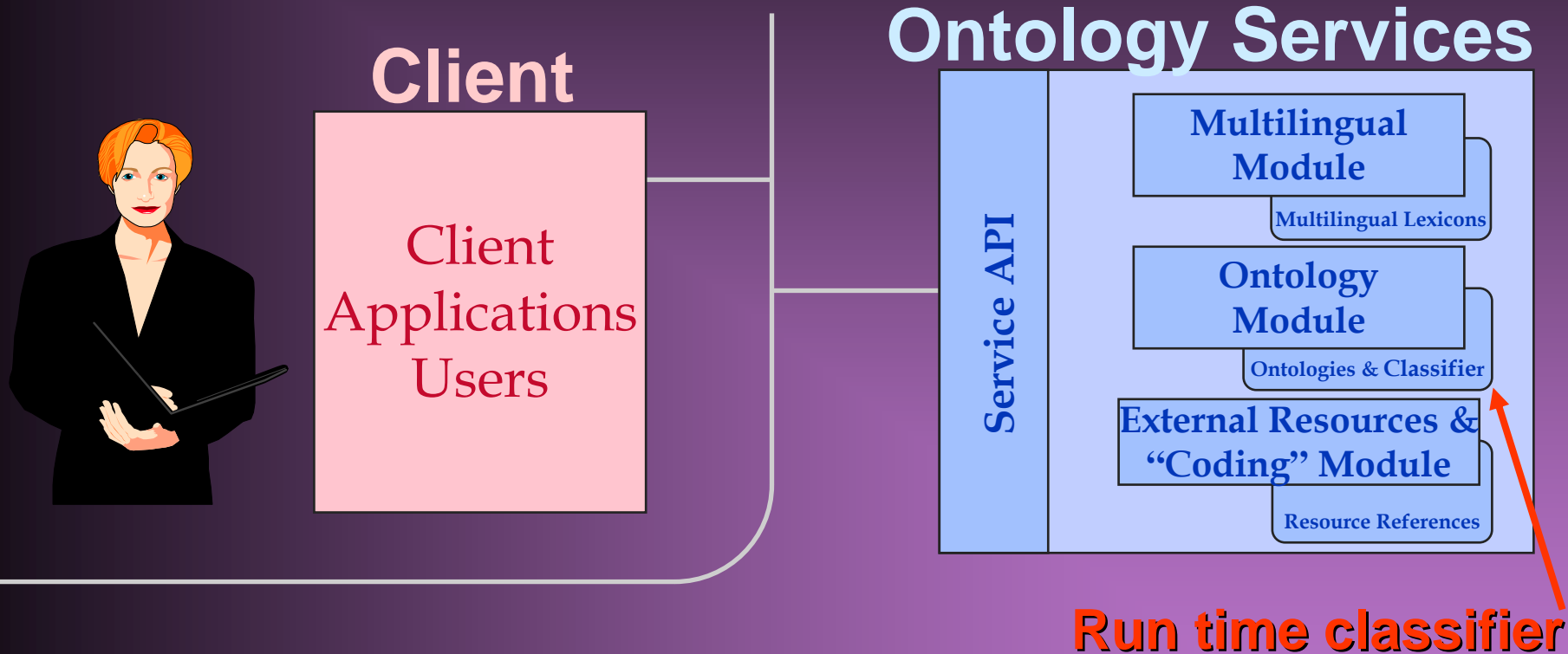
When to use a classifier 2: Post Coordination Classifier as an inference engine

- When the ontology too big – “Lazy classification” on demand



Big on the outside: small kernel on the inside

Often combined with other services: Example - the GALEN Server



...but... Why use a Classifier?

- To *compose* concepts
 - *Allow conceptual lego*
- To manage *polyhierarchies*
 - *Adding abstractions (“axes”) as needed*
 - *Normalisation*
 - *Untangling*
 - *labelling of “kinds of is-a”*
- To avoid *combinatorial explosions*
 - *Keep bicycles from exploding*
- To manage *context*
 - *Cross species, Cross disciplines, Cross studies*
- To check *consistency* and *help users find errors*

Logic-based Ontologies: Conceptual Lego



hand

extremity

body

chronic

acute

abnormal

normal

ischaemic

deletion

polymorphism

gene

protein

cell

expression

Lung

inflammation

infection

bacterial

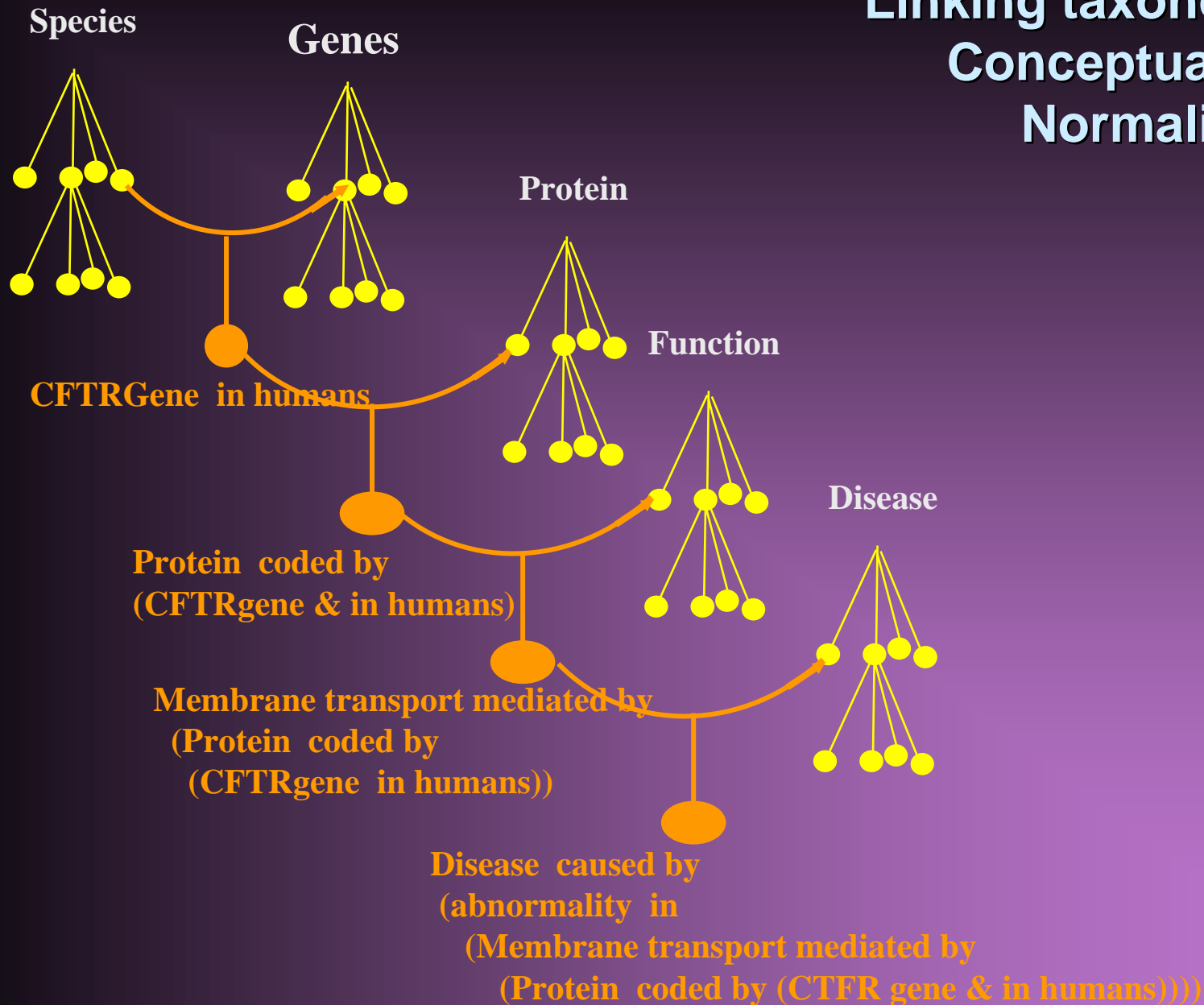
Logic-based Ontologies: Conceptual Lego

“SNPolymorphism of CFTRGene causing Defect in MembraneTransport of ChlorideIon causing Increase in Viscosity of Mucus in CysticFibrosis...”



“Hand which is anatomically normal”

Linking taxonomies: Conceptual Lego Normalisation



Logic Based Ontologies: The basics

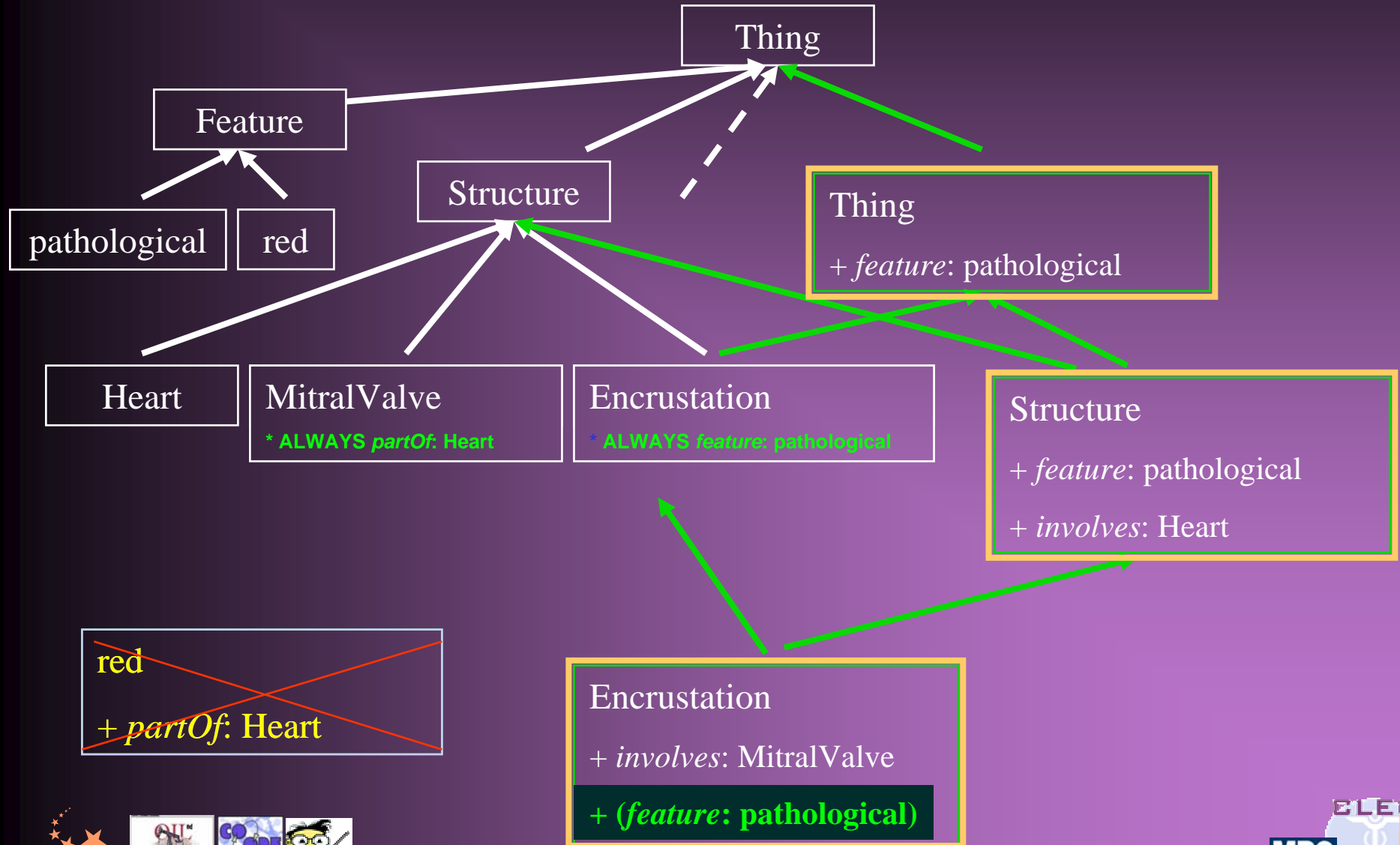
Primitives

Descriptions

Definitions

Reasoning

Validating
(constraining cross products)



Example demonstrations: Take a Few Simple Concepts & Properties

The image displays two windows from an ontology editor. The left window, titled "Asserted Hierarchy", shows a tree structure of classes. The root is "owl:Thing", which branches into "Domain_entity", "Gene", "Organism", "Process", "Protein", and "ValuePartition". "Domain_entity" further branches into "Disease", "Breast_cancer", "Cystic_fibrosis", "HIV", and "Sickle_cell_disease". "Breast_cancer" has a sub-class "BrCa_Breast_Cancer". "Gene" has sub-classes "BrCa_Gene", "CFTR_Gene", and "G152_Gene". "Organism" has sub-classes "Human" and "Mouse". "Process" has sub-classes "Apoptosis", "Oxygen_transport", and "Trans_Membrane_Ion_transport". "Protein" has a sub-class "Haemoglobin".

The right window, titled "Properties", shows a list of properties. The selected property is "has_normality_status". Other properties include "is_linked_to ↔ has_link_to", "Relation_property", "has_link_to ↔ is_linked_to", "causes_disease ↔ is_disease_caus...", "codes_for ↔ is_coded_for_by", and "has_function ↔ is_function_of".

Combine them in Descriptions which can be simple....

The screenshot displays a software interface for editing an ontology. On the left, a tree view shows a hierarchy starting with 'owl:Thing', followed by 'Domain_entity', 'Disease', and 'Sickle_cell_disease' (which is highlighted). Other categories include 'Gene', 'Organism', and 'Process'. On the right, the 'Sickle_cell_disease' class is selected, showing its 'rdfs:comment' field and a tabbed interface for 'Asserted Conditions'. The 'Asserted Conditions' tab is active, showing a list of conditions: 'Disease' and 'exists is_disease_caused_by Sickling_haemoglobin', both marked as 'NECESSARY'.

**Sickle cell disease is a disease caused
some sickling haemoglobin**

or which can be as complex as you like

The screenshot shows an ontology editor interface. On the left is a class hierarchy tree under 'owl:Thing', including 'Domain_entity', 'Disease', 'Breast_cancer', 'Cystic_fibrosis', 'Gene', and 'Organism'. The 'Cystic_fibrosis' class is highlighted. On the right, a text field contains 'Cystic_fibrosis' and 'rdfs:comment'. An 'Edit OWL Expression' dialog box is open, displaying the following OWL expression:

```
∃ is_disease_caused_by (Trans_Membrane_Ion_transport ∩  
  ∃ has_normality_status nonNormal ∩  
  ∃ is_function_of (Protein ∩  
    ∃ is_coded_for_by CFTR_Gene))
```

Cystic fibrosis is caused by some non-normal ion transport that is the function of a protein coded for by a CFTR gene

Add some definitions

The screenshot shows a web ontology editor interface. On the left, an 'Asserted Hierarchy' tree is displayed with the following structure:

- owl:Thing
 - Domain_entity
 - Disease
 - Breast_cancer
 - Cystic_fibrosis
 - Diseases_linked_to_CFTR_genes
 - Diseases_linked_to_Genes
 - HIV
 - Haemoglobinopathy
 - Sickle_cell_disease
 - Gene
 - BrCa_Gene
 - CFTR_Gene
 - G152_Gene
 - Organism
 - Human

The right-hand pane shows the definition for the selected class 'Diseases_linked_to_CFTR_genes'. The 'rdfs:comment' field is empty. Below, the 'Asserted Conditions' section shows a logical definition:

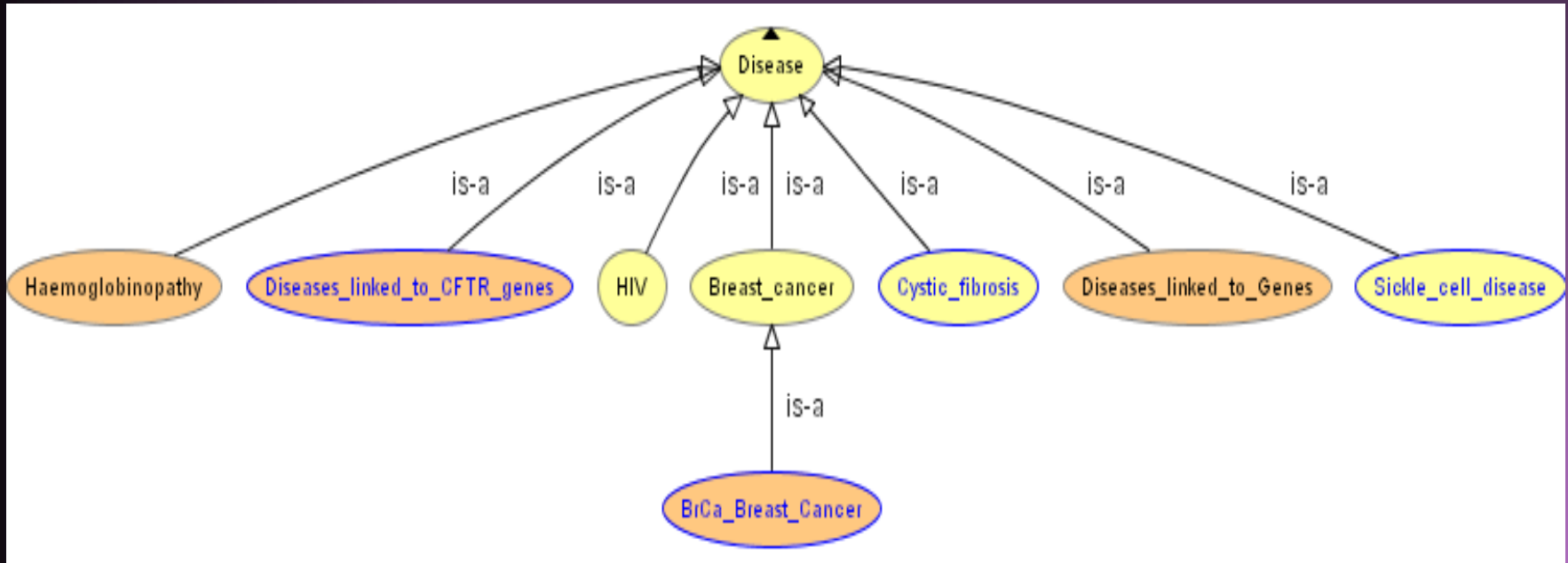
NECESSARY & SUFFICIENT

- Disease
- \exists is_linked_to CFTR_Gene

NECESSARY

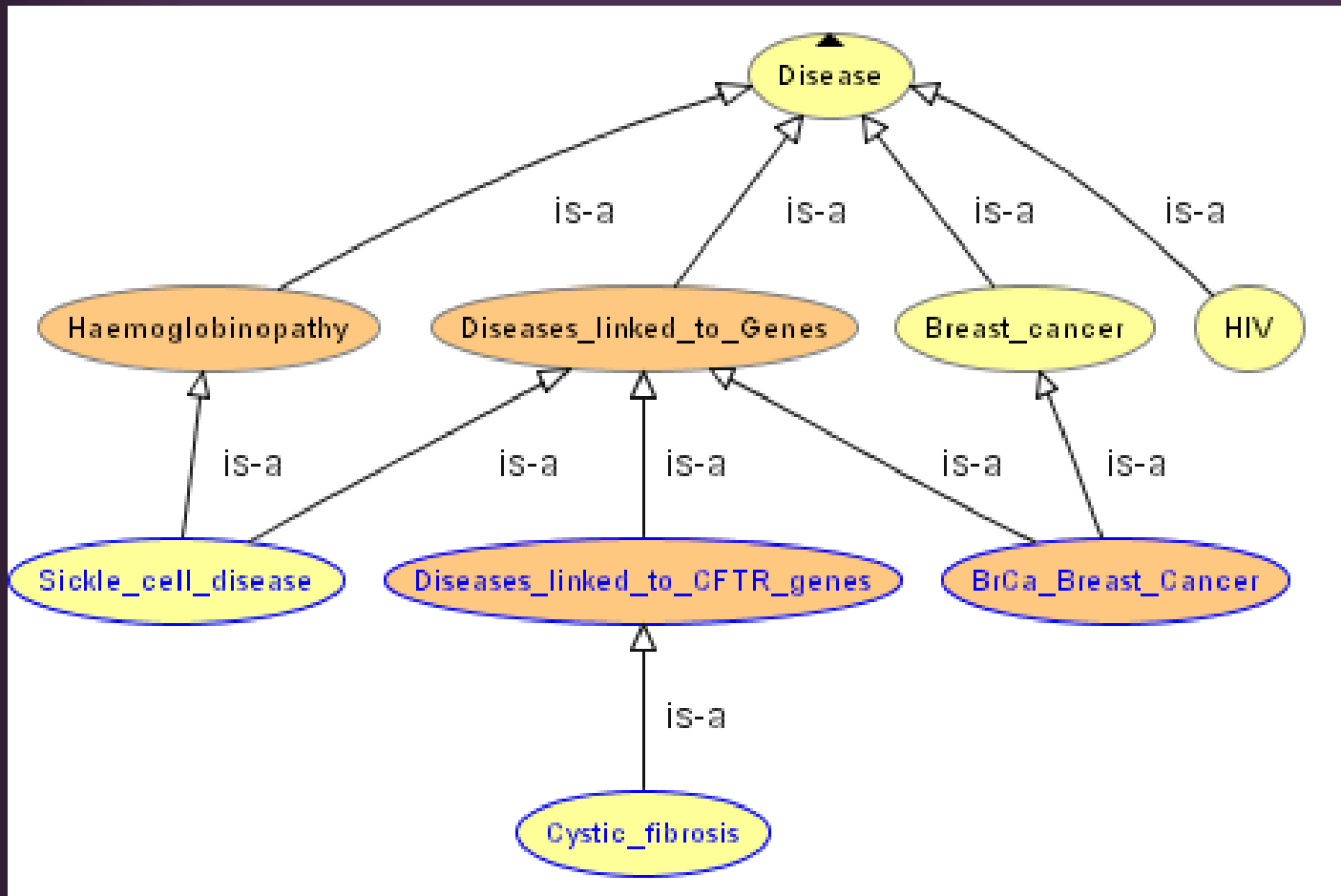
“Diseases linked to CFTR Genes”

We have built a simple tree

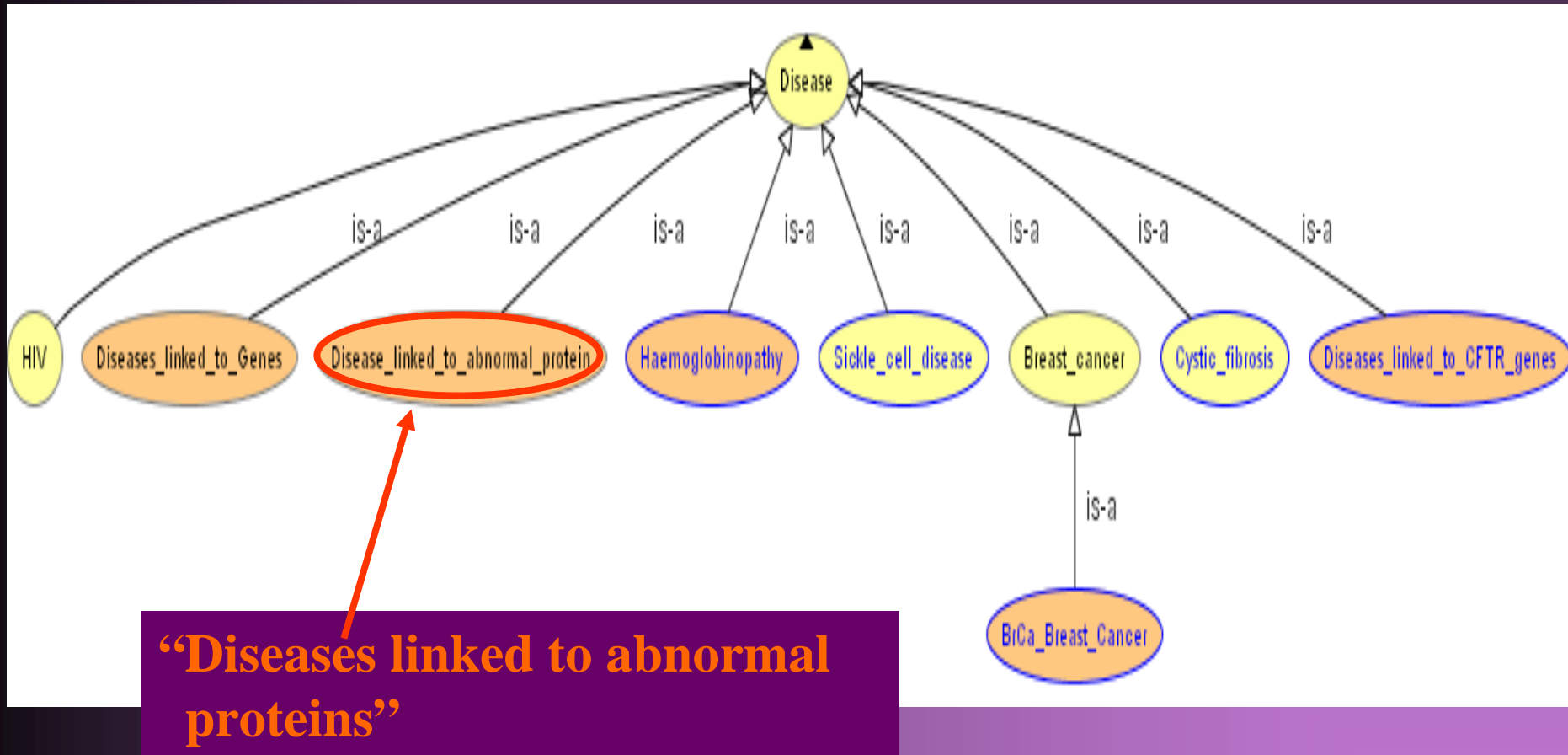


easy to maintain

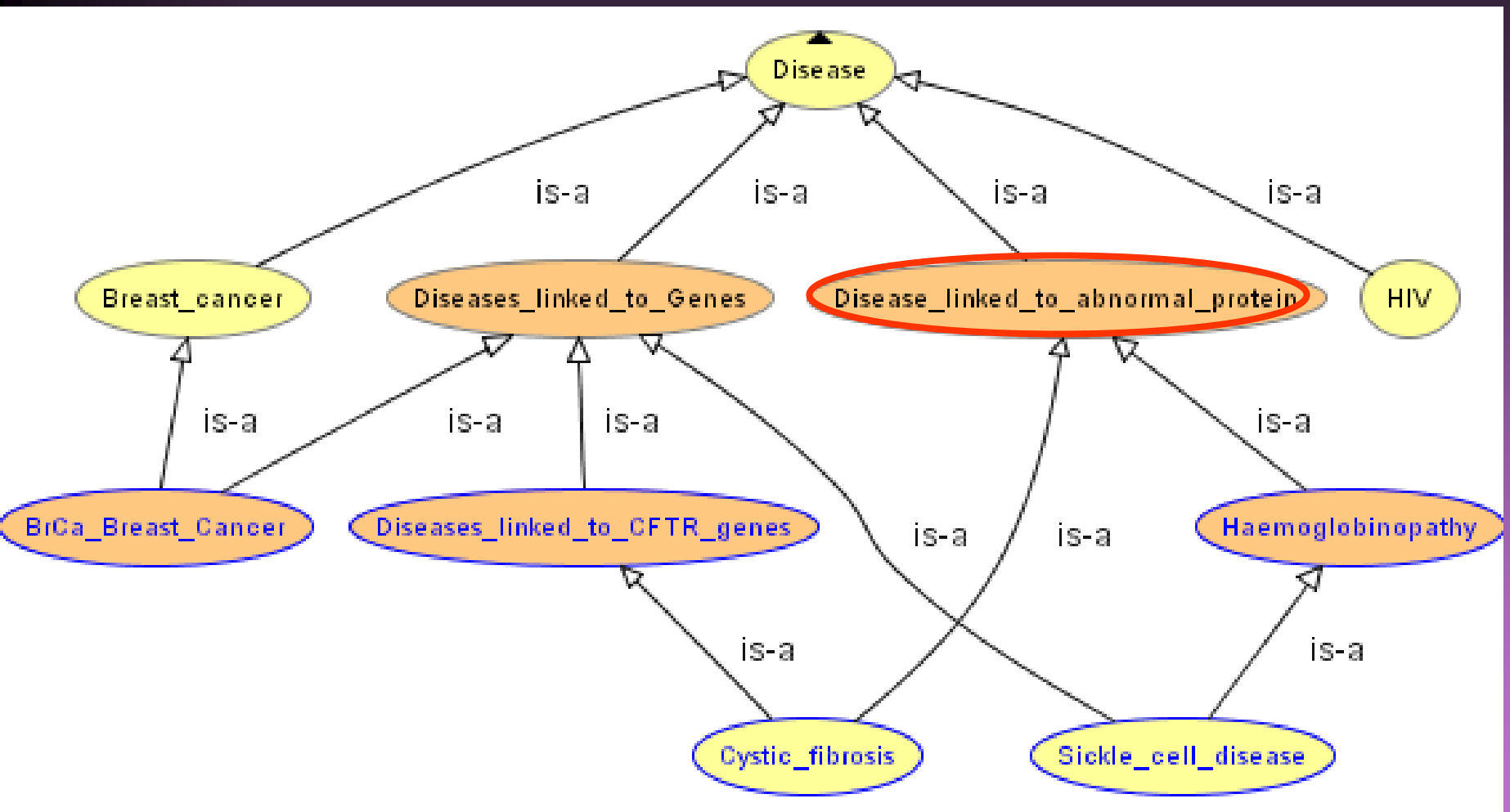
Let the classifier organise it



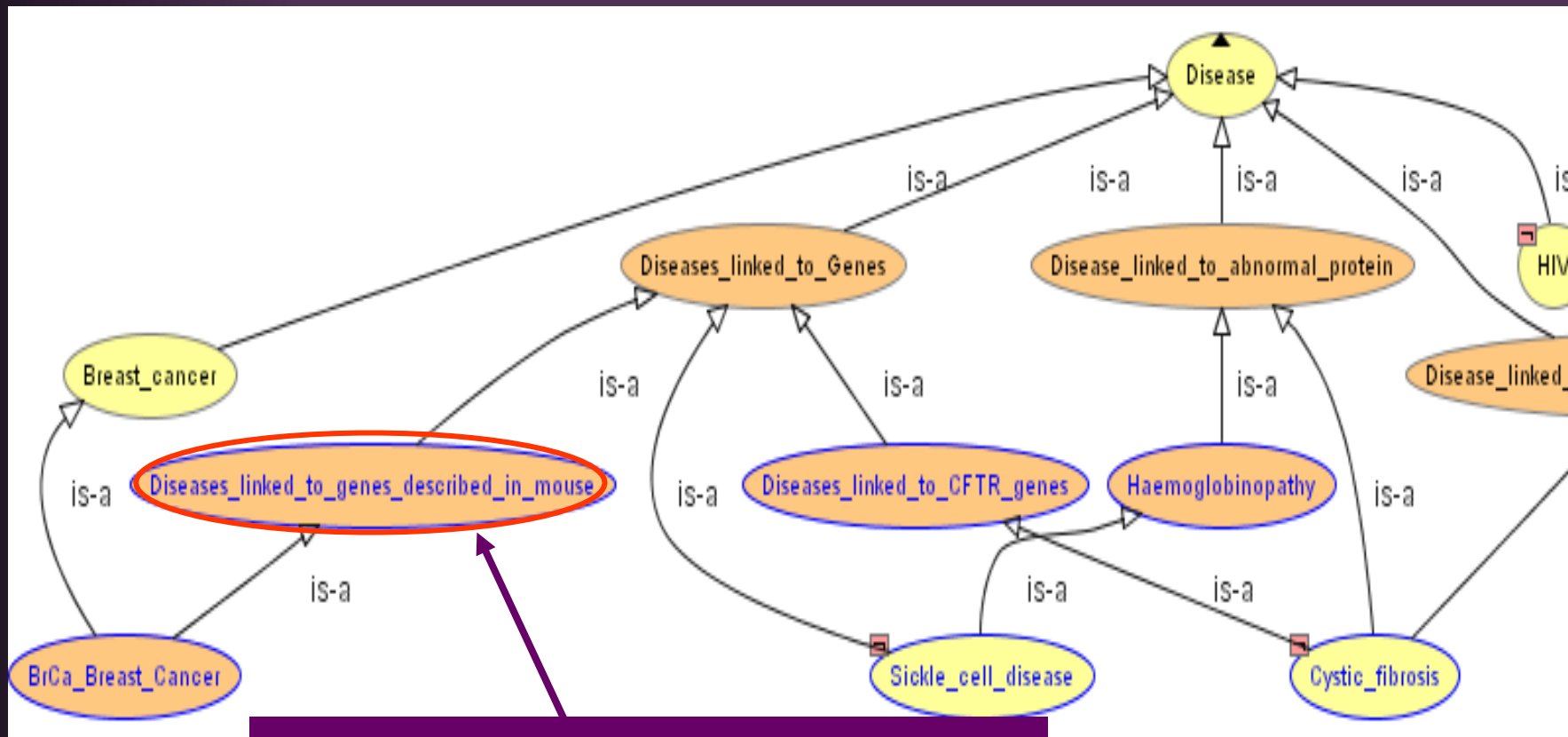
If you want more abstractions, just add new definitions (re-use existing data)



And let the classifier work again



And again – even for a quite different category



“Diseases linked genes described in the mouse”

And let classifier check consistency (My first try wasn't)

owl:Thing

- Domain_entity
 - Disease
 - Breast_cancer
 - Cystic_fibrosis**
 - Disease_linked_to_abr...
 - Disease_linked_to_abr...
 - Diseases_linked_to_G...
 - HIV
 - Gene
 - Organism
 - Process
 - Protein
 - ValuePartition

Cystic_fibrosis

rdfs:comment

Property

Asserted Inferred Abstract Syntax Abstract Syntax Abstract E

Asserted Conditions

NECESSARY & SUFFICIENT

NECESSARY

Disease

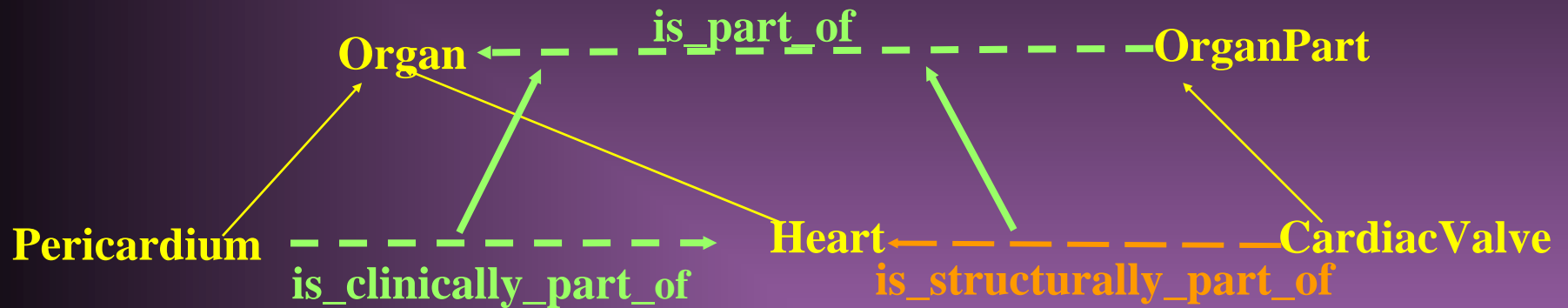
(∃ is_disease_caused_by Trans_Membrane_Ion_transport) ⊆ (∃ has_norm...)

The intersection of

- any object where some instances are disease c
- any object which has a non normal as its norma
- any object where some instances are function o
 - protein
 - any object which has a non normal as its no
 - any object where some instances are coded

Class	Changed superclasses
BrCa_Breast_Cancer	Added Diseases_linked_to_genes_described_in_mouse
Cystic_fibrosis	Inconsistent
Diseases_linked_to_CFTR_ge...	Moved from Disease to Diseases_linked_to_Genes
Diseases_linked_to_genes_de...	Moved from Disease to Diseases_linked_to_Genes
Haemoglobinopathy	Moved from Disease to Disease_linked_to_abnormal_protein

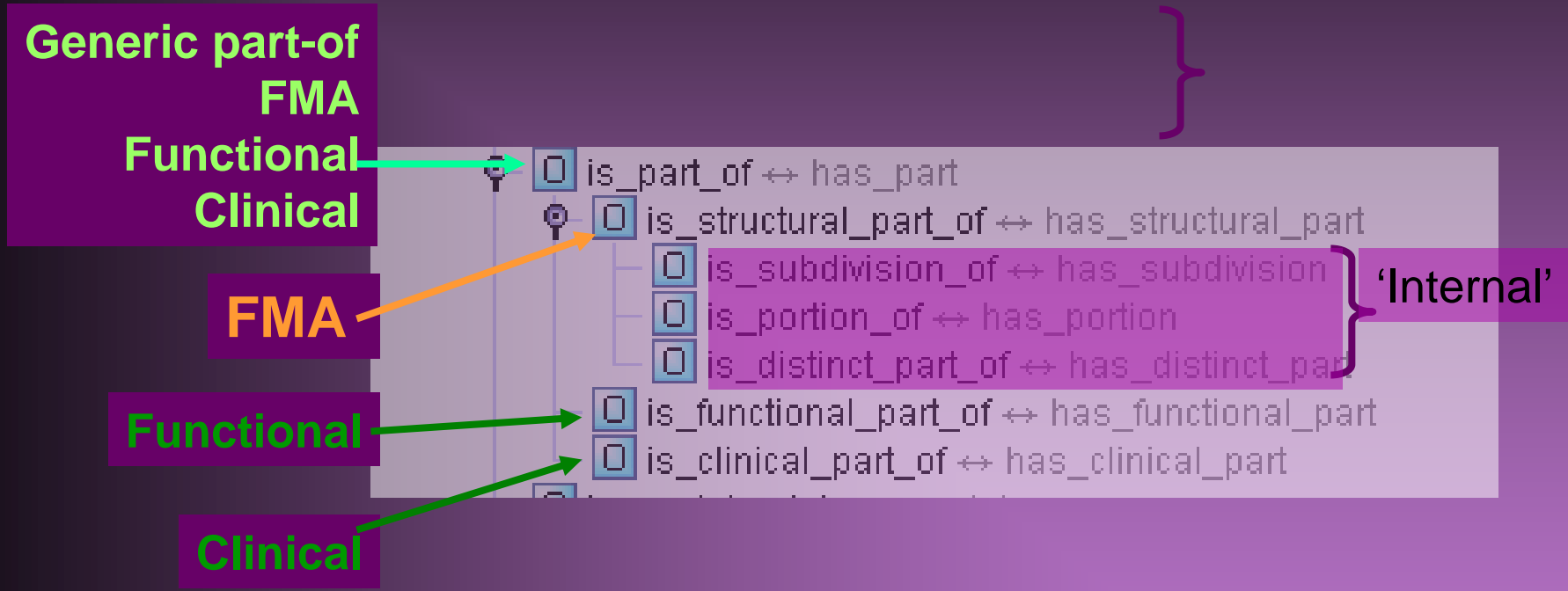
Represent context and views by variant properties



Disease of (Heart or part-of-heart)

**Disease of
Pericardium**

Integration of Contexts in Protégé-OWL



Protégé-OWL alternative views

Disorder of “Clinical heart”

“Disorder of heart of any part of the heart”
(including *clinical* and *functional* parts)

NECESSARY & SUFFICIENT

Disorder

\exists has_locus (Heart \sqcup (\exists is_part_of Heart))

NECESSARY

Disorder of “FMA heart”

“Disorder of heart or any *structural* part of the heart”

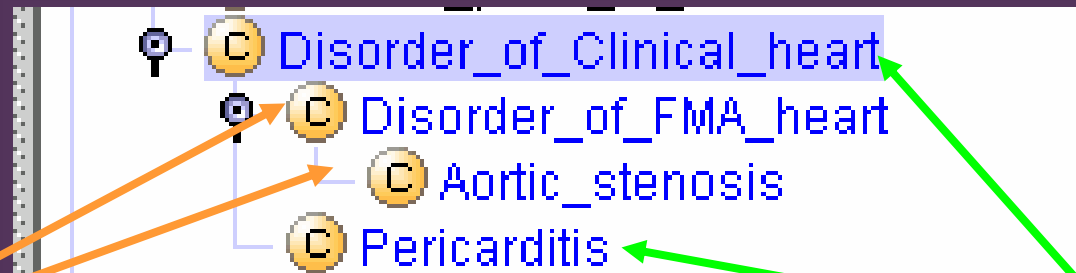
NECESSARY & SUFFICIENT

Disorder

\exists has_locus (Heart \sqcup (\exists is_structural_part_of Heart))

NECESSARY

Consequences for classification of diseases



Disorders of the things anatomists recognise as parts of the heart

Doctors consider it a heart disease, even though developmentally/structurally it is not

Summary: Why Classify?

- To *compose* concepts
- To untangle *polyhierarchies* – to *Normalise*
- To avoid *combinatorial explosions*
- To manage *context*
- To check *consistency* and *help users find errors*

Summary: When to Classify?

Applications do not need a classifier to benefit from classification

- **Pre-coordination**
 - **If concepts/terms can be predicted**
 - **When classifier is not available at run time**
 - **When we must fit with legacy applications**
- **Post-coordination**
 - **When a a few concepts are needed from a large potential set**
 - **When a classifier is available**
 - **and time cost is acceptable**
 - **When applications can be built or adapted to take advantage**

Remember: Think about the Life Cycle

- The life cycle for pre-coordinated ontologies
 - Gather requirements, sketch, experiment
 - Establish patterns – design a “language”
 - What a subject domain expert can learn in a few days
 - Bulk authoring
 - Classification
 - Quality assurance
 - Commit classifier results to a pre-coordinated ontology & deliver
 - Taxonomies (Protégé, DAG-Edit, OWL-Lite, RDF(S), Topic Maps, ...
 - Query and use with you favourite tool

Development & evolution