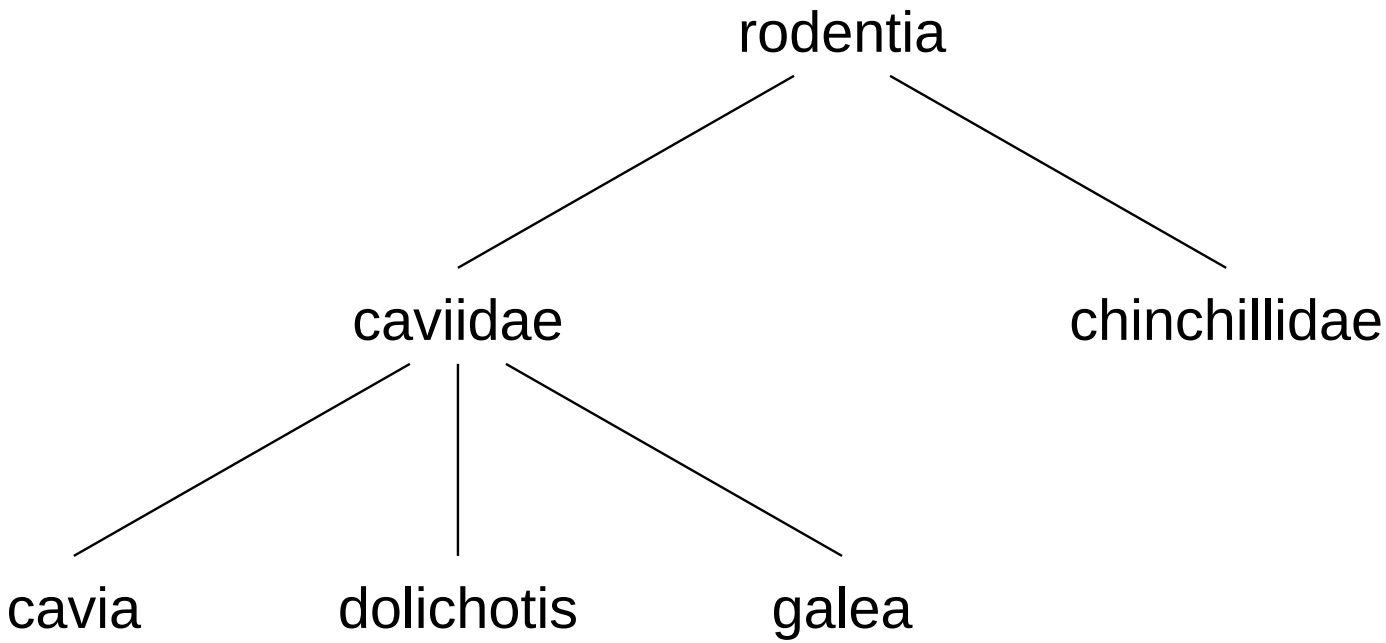
Using Ontologies to Derive Transitive Queries

Graham J.L. Kemp
Department of Computing Science
Chalmers University of Technology,
SE-412 96 Göteborg
Sweden

kemp@cs.chalmers.se

http://www.cs.chalmers.se/~kemp/

Reasoning with ontologies when answering queries (1)



If a user asks for information about samples where the source is identified by the taxonomy term ‘‘rodentia’’, then we want the system also to return information on samples where the recorded taxonomy term is below ‘‘rodentia’’ in the hierarchical vocabulary.

Reasoning with ontologies when answering queries (2)

1. Analyse query

   Is a value for a string-valued attribute specified in the query?

2. Check database metadata

   Are the values stored for that attribute chosen from a hierarchical vocabulary?

3. Find alternative acceptable values

   If ‘‘yes’’ to both of the above questions, find the set of more specialised values for this attribute that would also satisfy the query by examining the sub-tree of terms below the given term.

Reasoning with ontologies when answering queries (3)

Extract from schema of database module storing the taxonomy:

```
declare taxonomy_entry ->> entity
declare name(taxonomy_entry) -> string
declare is_of_taxa(taxonomy_entry) -> taxonomy_entry
```

and derived functions:

```
define acceptable_alternative_names(s in string) ->> string
  name(self_and_more_specialised(t in taxonomy_entry such that
                                   name(t)=s));


define self_and_more_specialised(t in taxonomy_entry) ->>
         taxonomy_entry
  ({t} union self_and_more_specialised(is_of_taxa_inv(t)));
```

Reasoning with ontologies when answering queries (4)

Extract from a P/FDM implementation of the MIAME schema:

```
declare sample                          ->> entity
  declare id(sample)                     -> string
  declare organism_ncbi(sample)          -> string
  declare treatment_protocol(sample)     -> string
  declare used_in_experiment(sample)     -> experiment
  declare sample_parameters(sample)      ->> parameter
key_of sample is id
```

Daplex query:

```
for each s in sample such that organism_ncbi(s)="rodentia"
  print(id(s),treatment_protocol(s));
```

The system returns information about all samples where the recorded taxonomy name is “rodentia” **or below**.

Daplex schema for GO terms and relationships

```
declare go_term                ->> entity
  declare name(go_term)         -> string
  declare term_type(go_term)    -> string
  declare isa(go_term)         ->> go_term
  declare partof(go_term)      ->> go_term
  key_of go_term is name, term_type;
```

The functions *isa* and *partof* are multi-valued (indicated by the double-headed arrows), as are the inverse relationships (*isa_inv* and *partof_inv*) that are maintained automatically by the P/FDM system.

A GO term is not uniquely identifi ed by its name. For example, “B-cell receptor” is both the name of a function and the name of a component, and “tubulin folding” is both the name of a function and the name of a process.

GO usage: logical relations

[http://www.geneontology.org/doc/GO.usage.html]

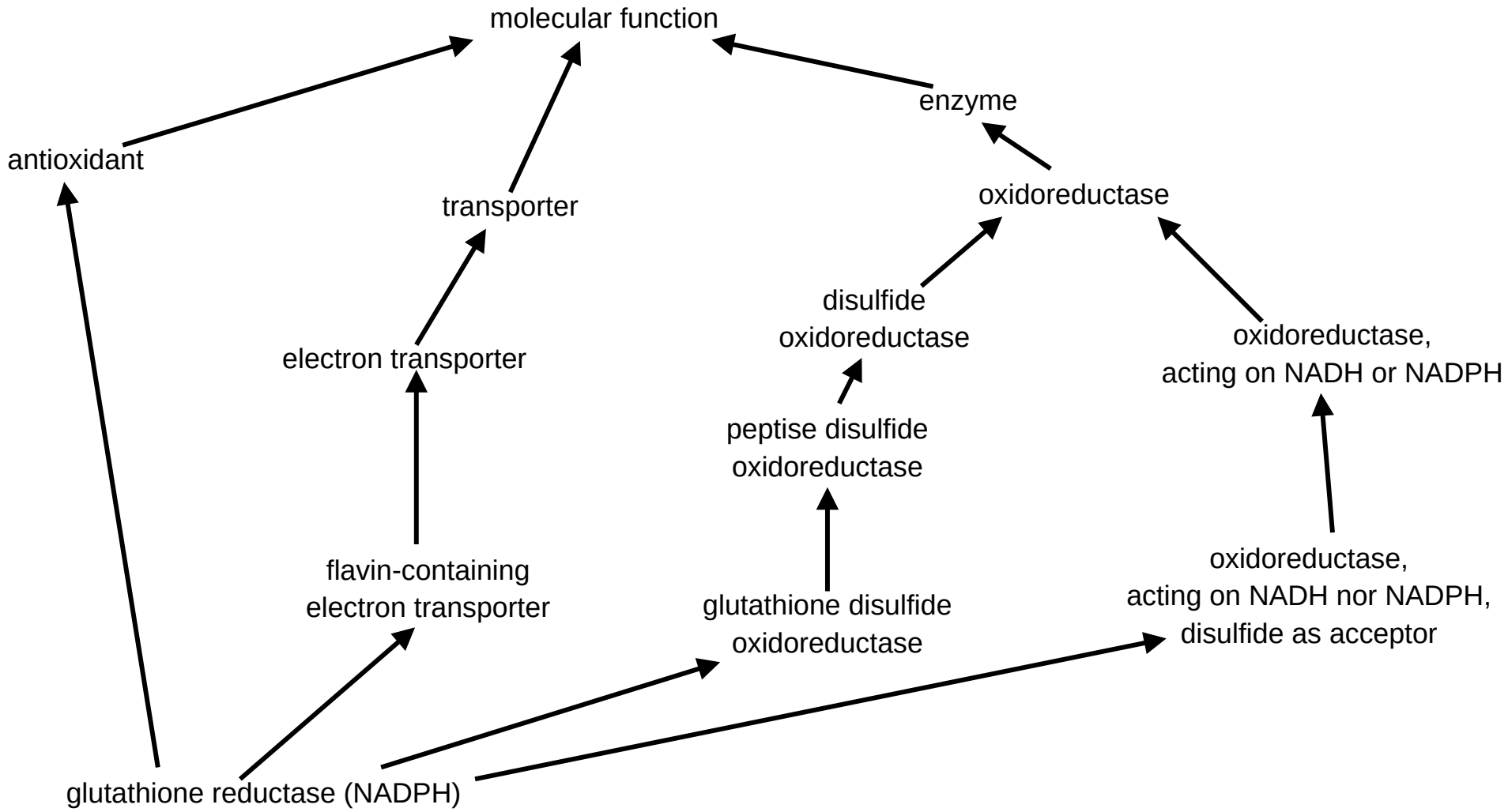
```
(1)     if A is part of B
        and C is an instance of B,
        is A part of C? --YES

(2)     if A is an instance of B
        and B is an instance of C,
        is A is an instance of C? --YES

(3)     if A is part of B
        and B is part of C,
        is A part of C? --YES

(4)     if A is an instance of B
        and C is part of A,
        is C part of B? --NOT NECESSARILY
```

GO usage: logical relations

If terms A and B are directly related and terms B and C are directly related then can we infer a relationship between terms A and C?

	B ISA C	C ISA B	B PARTOF C	C PARTOF B
A ISA B	A ISA C			C PARTOF A
B ISA A		C ISA A		
A PARTOF B		A PARTOF C	A PARTOF C	
B PARTOF A				C PARTOF A

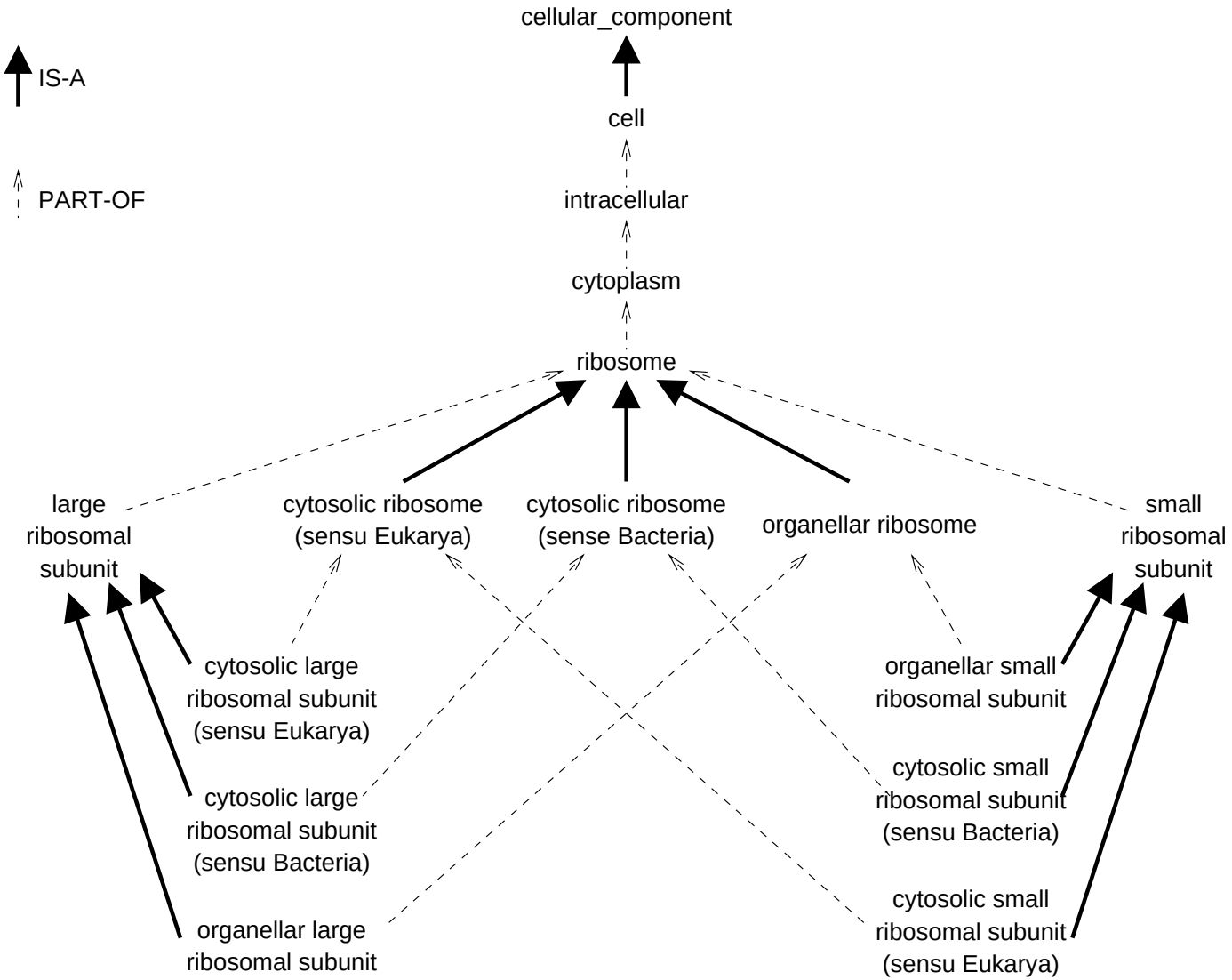
GO: function example



Finding all more specialised terms

```
define all_more_specialised_terms(t in go_term) ->> go_term
  (isa_inv(t) union all_more_specialised_terms(isa_inv(t)));
```

GO: component example



Finding all sub-parts

```
define all_subparts(t in go_term) ->> go_term
        in tempmod
  ((partof_inv(t)
     union all_subparts(partof_inv(t)))
     union all_subparts(isa(t)));
```

Example query:

```
|: for the t in go_term such that
|:     name(t) = "organellar ribosome" and
|:     term_type(t) = "component"
|:   print(name(all_subparts(t)));

organellar small ribosomal subunit
organellar large ribosomal subunit
small ribosomal subunit
large ribosomal subunit
```

Dependent ontology terms

[http://www.geneontology.org/doc/GO.usage.html]

> If either "X biosynthesis" or "X catabolism" exists,
> then the parent "X metabolism" must also exist.

Species-specific terms

```
declare go_term                      ->> entity
  declare name(go_term)               -> string
  declare term_type(go_term)          -> string
  declare taxonomy_entry(go_term)  -> taxonomy_entry
  declare isa(go_term)               ->> go_term
  declare partof(go_term)            ->> go_term
  key_of go_term is name, term_type key_of(taxonomy_entry);
```