(LOGICAL) DOMAIN MODELLING

- Provision of a formal, in particular logical language for knowledge representation.
- Application of these means to represent the formal structure and the general facts of an application domain.
- Application of the formal language to represent concrete application situations (individuals, instances).

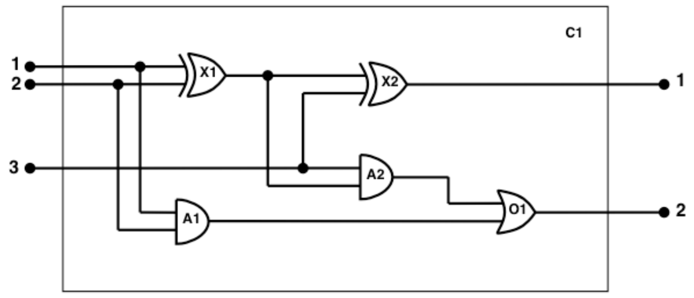
The construction of a knowlege base ("knowledge engineering") comprises:

- Determination of the domain
- Determination of the domain concepts
- Representation of the objects and relations in the domain.

Domain Modelling: An Example

- We begin by studying an example of modelling a simple domain with First Order Logic.
- We assume some familiarity with standard FOL; more general issues will be taken up in the next chapter.
- Here we ask how a modelling task can be achieved with FOL, and
- which requirements modelling tasks impose on logical languages.
- Furthermore, we outline a general framework for formal ontologies.

Example Domain: Electronic Circuits (Russell/Norvig)



Modelling Steps

1. What's the subject matter? Which tasks shall be solved with the help of the domain model?
2. Putting down the domain vocabulary: predicates, functions, relations, and constants $\Rightarrow$ terminological framework of the "formal ontology".
3. Expressing general rules which hold in the domain. If necessary, revise the domain terminology in order to express the rules clearly and concisely and to keep the set of rules rather small. Rules correspond to (material) axioms, by means of which the inference procedure can derive statements about the domain.
4. Constructing an actual situation description with the creation of instances.
5. Posing queries to the inference procedure.

Questions During the Modelling Process

For the statements and rules:

- What are the validity or truth conditions?
- Which facts meet a particular statement?
- Generality?
- Potential for generalization?
- Relationships and connections, in particular among super- and subclasses?

Example: Electronic Circuits
1. Identify the Task and Assemble the Relevant Knowledge

Digital circuits consist of wires (connections) and gates. Signals flow along wires to the input terminals of gates, and each gate produces a signal on the output terminal that flows along another wire.

There are four kinds of gates: AND, OR, XOR with two terminals and NOT with one. All gates have exactly one output terminal. Circuits assembled from gates, have input and output terminals.

Goal: Analysis of designs — *Do they meet their specification?*

We have to represent:

circuits, their *terminals* and the corresponding *signals*.

To represent the signals we need representations of particular *gates* and of *types of gates*. These objects suffice to represent the connectivity of the terminals.

Nothing else has to be considered for the given goal — but for other goals, e.g. error diagnosis.

2. Decide on the Domain Vocabulary

- Distinguish a gate from other gates: Naming with constants X_1, $X_2, \ldots$
- Types of gates: typechecking-function $Type(X_1) = XOR$ or -predicate $Type(X_1, XOR)$
- Terminals: constants, or, better: functions $Out(1, X_1)$, etc.
- Connectivity: predicate $Connected$, e.g., $Connected(Out(1, X_1), In(1, X_2))$
- Signals: function $Signal(Terminal)$, two-valued: $1, 0$

3. Encode General Knowledge of the Domain by Rules

In our domain seven rules are sufficient:

1. If two terminals are connected, then they have the same signal
   $\bigwedge_{t_1,t_2} .Connected(t_1,t_2) \rightarrow Signal(t_1) = Signal(t_2)$.
2. The signal at every terminal is either on or off:
   $\bigwedge_t .Signal(t) = 1 \vee Signal(t) = 0$.
   $1 \neq 0$
3. OR gate: output signal is on iff any of its input signals is on
   $\bigwedge_g .Type(g) = OR \rightarrow (Signal(Out(1,g)) = 1 \leftrightarrow \bigvee_n .Signal(In(n,g)) = 1.)$.
4. AND gate: output signal is off iff any of its input signals is off
   $\bigwedge_g .Type(g) = AND \rightarrow (Signal(Out(1,g)) = 0 \leftrightarrow \bigvee_n .Signal(In(n,g)) = 0.)$.

5. XOR gate: output signal is on iff its input signals are different
   $\bigwedge_g .Type(g) = XOR \rightarrow (Signal(Out(1,g)) = 1 \leftrightarrow Signal(In(1,g)) \neq Signal(In(2,g)))$.
6. NOT gate: output signal is complementary to the input signal
   $\bigwedge_g .Type(g) = NOT \rightarrow Signal(Out(1,g)) \neq Signal(In(1,g))$.

4. Instantiation: Specific Problem Instance

Example: cf. figure

$Type(X_1) = XOR$	$Type(X_2) = XOR$
$Type(A_1) = AND$	$Type(A_2) = AND$
$Type(O_1) = OR$	
$Connected(Out(1,X_1), In(1,X_2))$	$Connected(In(1,C_1), In(1,X_1))$
$Connected(Out(1,X_1), In(2,A_2))$	$Connected(In(1,C_1), In(1,A_1))$
$Connected(Out(1,A_2), In(1,O_1))$	$Connected(In(2,C_1), In(2,X_1))$
$Connected(Out(1,A_1), In(2,O_2))$	$Connected(In(2,C_1), In(2,A_1))$
$Connected(Out(1,X_2), Out(1,C_1))$	$Connected(In(3,C_1), In(2,X_2))$
$Connected(Out(1,O_1), Out(2,C_1))$	$Connected(In(3,C_1), In(1,A_2))$

5. Pose Queries to the Inference Procedure

- What combination of inputs would cause the first output of C_1 (the sum bit) to be off and the second output of C_1 (the carry bit) to be on?
  Answer:
  $(i_1 = 1 \wedge i_2 = 1 \wedge i_3 = 0) \vee$
  $(i_1 = 1 \wedge i_2 = 0 \wedge i_3 = 1) \vee$
  $(i_1 = 0 \wedge i_2 = 1 \wedge i_3 = 1)$
- What are the possible sets of values of all the terminals for the adder circuit?
  $\bigvee_{i_1,i_2,i_3,o_1,o_2} .Signal(In(1,C_1)) = i_1 \wedge Signal(In(2,C_1)) = i_2 \wedge Signal(In(3,C_1)) = i_3 \wedge Signal(Out(1,C_1)) = o_1 \wedge Signal(Out(2,C_1)) = o_2$.
  Answer:
  Complete input-output table for the device — can be used to check that it does in fact add its inputs correctly: **circuit verification**.

Formal Ontologies

Formal ontologies consist of (formal) definitions ("descriptions") of the concepts and relations in a domain:

- The **concepts** (also: classes, categories) result from predication and abstraction, represented by predicates.
  Question: *What?* — "substantia" (as opposed to function: *How?*).
- Relations between concepts result from predicator rules and are represented in a super-/sub-concept hierarchy: hyponymy ("divisio"); **is**-relation
- Concepts are assigned **properties** ("roles", attributes), to denote "accidentia", represented by (binary) **relations**.
  has-relation (as opposed to part-whole — mereonymy, "partitio")

- Roles may be restricted: **constraints**
- Further (content-based) relations between concepts (constraints) are laid down by rules ("axioms").
- Individuals: **Instances** of classes (tokens of types).

A formal ontology (of a domain) defines, how and about which objects, substances, aggregates, changes, events, actions, time and place specifications, etc. can be "spoken".

Modelling Decisions: Concepts and Roles

- What shall be represented as a concept, what as a property (role)?
- Disjointness of concepts; Partitions (disjoint exhaustive partitions)
- Taxonomic hierarchy (super-/sub-concept relations);
  genera and species ("differentia specifica"); role hierarchy
- Inheritance of properties
- Reification of concepts: predicates become objects of the language in order to state assertions about the concept proper — not about its instances (extension).
- Typicality of instances and "natural kinds" — prototypes (are instances!) vs. classes
- The problem of defaults

Generalizations of Formal Domain Ontologies

Special formal domain ontologies often make simplifying assumptions (cf. **example**):

- Time is not represented;
- Signals are fixed and the propagation of signals is not represented;
- The structure of the circuits remains constant.

Generalizations:

- Consider signals at particular time points, take into account length of connections and delays during propagation in connections and gates
  $\Rightarrow$ simulation of temporal circuit properties

- Consider spontaneous changes in the circuits' structure or the gates' properties
  ⇒ error diagnosis, investigations of reliability
- Transition from the topological connection structure to a realistic geometric representation of the layout
  ⇒ electromagnetical effects.

General Formal Ontologies

Can we achieve a convergence to general formal ontologies by generalizations of special formal domain ontologies??

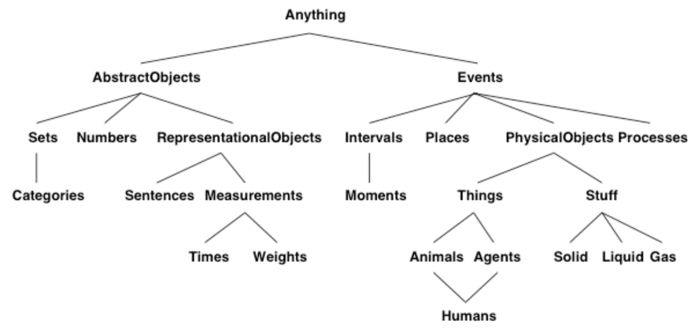
Each general ontology should — in combination with domain specific axioms — be applicable in special domains; in nontrivial applications often a **union** of different domain models is necessary (simultaneous use for problem solving).

Components of general formal ontologies

- Concepts (also: categories, classes, types)
- Physical objects, substances
- Abstract objects

- Composite objects, part-whole relations (mereonymic hierarchies)
- Time, space, change; measures
- Events and processes
- Beliefs, "mental" objects

Example of a "Top Level" Ontology (Russell/Norvig)



Reference and Application Ontologies

. . . a further generalization step

Reference ontologies

- Generic, universal conceptual inventory
  Representation language and fundamental distinctions
- Foundational relations
  parts and wholes (mereonymy), similarity, dependence, connection, inherence, temporal order

Application ontologies

- Modelling particular application domains

Reference Ontologies

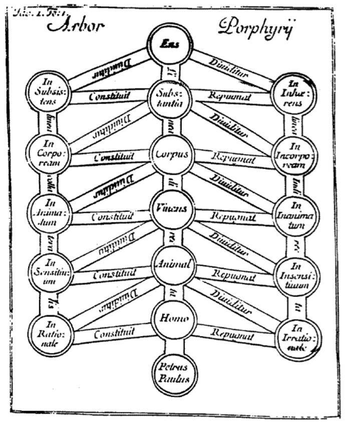
- Characteristics
  - Theoretical focus on (axiomatic) representation
  - Methodological emphasis on accuracy and comprehensiveness
  - Metaclass schema
- Use
  - General theoretical framework
  - Generic upper ontology (as far as computationally tractable)
- Is formal reconstruction independent of goals?

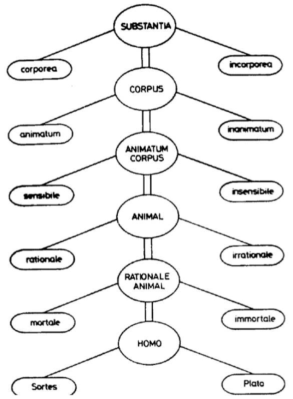
Problems: Merging ontologies; connecting general with domain specific ontologies

Building / Selecting a “Top Level” Ontology

- Approaches rooted in philosophy
  - Aristotle’s category system:
    substance, quantity, quality, relation, where or when or being-in-a-position or having or doing or being-affected
    and Porphyrean trees;
  - Kant’s category table; Brentano; Sowa;
  - Methodologically “clean” formal ontologies: DOLCE (Guarino)
- Approaches rooted in a disciplinary view — according to specific needs —, e.g. computational linguistics
- Knowledge engineering approaches, e.g. SUMO (Standard Upper Merged Ontology), CYC: inter-operability!

Porphyrius’ Tree





SUMO as a Generic Base Model

"ISA-Hierarchy" of basal concepts: Upper levels of SUMO (Standard Upper Merged Ontology) — a pragmatic, empiric approach

```
Entity
  Physical
    Object
      SelfConnectedObject
        Region
        Substance
        CorpuscularObject
      Collection
    Process
  Abstract
    Class
      Set
    Relation
    Proposition
    Quantity
      Number
      PhysicalQuantity
    Graph
    Attribute
```

DOLCE (Guarino et al.): A Systematic Approach to the construction and Remodelling of Ontologies

. . . just a few remarks

Re-visiting conceptual modelling constructs

- Instantiation
- Generalization
- Association
- Aggregation

E.g.:

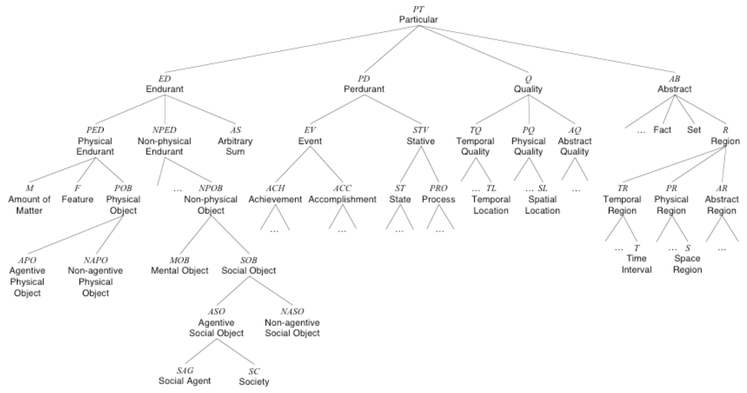
particulars and universals;
instance-of vs. membership;
subsumption vs. instantiation, composition, disjunction, polysemy, constitution;
identity;
part-of vs. part-whole relations;
attributes vs. arbitrary relations.

DOLCE: a Descriptive Ontology for Linguistic and Cognitive Engineering

Instead of a single upper level, provide a small set of *foundational ontologies*

- A first reference module for the foundational ontology library
- Strong cognitive bias (perception, culture, social conventions)
- Rich axiomatization
- Categories as "conceptual containers"

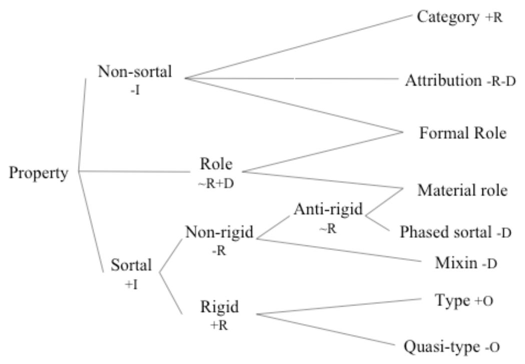
DOLCE: Basic Categories (Guarino)



DOLCE: Basic Relations (Guarino)

- Parthood
  - Between quality regions (immediate)
  - Between arbitrary objects (temporary)
- Dependence
  - Specific/generic constant dependence
- Constitution
- Inherence (between a quality and its host)
- Quale
  - Between a quality and its region (immediate, for unchanging ent)
  - Between a quality and its region (temporary, for changing ent)
- Participation
- Representation

DOLCE: Properties (Guarino)



Representation Languages for Formal Ontologies

Frame Languages (Fikes)

- Object-oriented representation languages
- Class–subclass taxonomies
- Prototype descriptions of class instances
- Frame systems perform standard set of inferences
  - Inheritance of attribute values and constraints
  - Type checking of attribute values
  - Checking number of attribute values
- Open Knowledge Base Connectivity (OKBC)

Later: Rephrasing in **Description Logics** ?!?