9.

Structured Descriptions

From sentences to objects

As we saw with frames, it useful to shift the focus away from the true *sentences* of an application towards the categories of *objects* in the application and their properties.

In frame systems, this was done *procedurally,* and we concentrated on hierarchies of frames as a way of organizing collections of procedures.

In this section, we look at the categories of objects themselves:

- objects are members of multiple categories
  e.g. a doctor, a wife, a mother of two
- categories of objects can be more or less specific than others
  e.g. a doctor, a professional, a surgeon
- categories of objects can have parts, sometimes in multiples
  e.g. books have titles, tables have legs
- the relation among the parts of an object can be critical in its being a member of a category
  e.g. a stack vs. a pile of bricks

Noun phrases

In FOL, all categories and properties of objects are represented by atomic predicates.

- In some cases, these correspond to simple *nouns* in English such as Person or City.
- In other cases, the predicates seem to be more like *noun phrases* such as MarriedPerson or CanadianCity or AnimalWithFourLegs.

Intuitively, these predicates have an internal structure and connections to other predicates.

e.g. A married person must be a person.

These connections hold by *definition* (by virtue of what the predicates themselves mean), not by virtue of the facts we believe about the world.

In FOL, there is no way to break apart a predicate to see how it is formed from other predicates.

Here we will examine a logic that allows us to have both atomic and non-atomic predicates: a description logic

Concepts, roles, constants

In a description logic, there are sentences that will be true or false (as in FOL).

In addition, there are three sorts of expressions that act like nouns and noun phrases in English:

- concepts are like category nouns — Dog, Teenager, GraduateStudent
- roles are like relational nouns — :Age, :Parent, :AreaOfStudy
- constants are like proper nouns — johnSmith, chair128

These correspond to unary predicates, binary predicates and constants (respectively) in FOL.

See also: generic frames, slots, and individual frames.
However: roles can have multiple fillers.

However, unlike in FOL, concepts need not be atomic and can have semantic relationships to each other.

roles will remain atomic (for now)

The symbols of DL

Three types of non-logical symbols:

- atomic concepts:

  Dog, Teenager, GraduateStudent

  We include a distinguished concept: Thing
- roles: (all are atomic)

  :Age, :Parent, :AreaOfStudy
- constants:

  johnSmith, chair128

Four types of logical symbols:

- punctuation: [,], (,)
- positive integers: 1, 2, 3, ...
- concept-forming operators: `ALL`, `EXISTS`, `FILLS`, `AND`
- connectives: $\sqsubseteq$, $\doteq$, and $\rightarrow$

The syntax of DL

The set of concepts is the least set satisfying:

- Every atomic concept is a concept.
- If r is a role and d is a concept, then [`ALL` r d] is a concept.
- If r is a role and n is an integer, then [`EXISTS` n r] is a concept.
- If r is a role and c is a constant, then [`FILLS` r c] is a concept.
- If d_1, ..., d_k are concepts, then so is [`AND` d_1, ..., d_k].

Three types of sentences in DL:

- If d and e are concepts, then ($d \sqsubseteq e$) is a sentence.
- if d and e are concepts, then ($d \doteq e$) is a sentence.
- If d is a concept and c is a constant, then ($c \rightarrow d$) is a sentence.

The meaning of concepts

Constants stand for individuals, concepts for sets of individuals, and roles for binary relations.

The meaning of a complex concept is derived from the meaning of its parts the same way a noun phrases is:

- [EXISTS n r] describes those individuals that stand in relation r to at least n other individuals
- [FILLS r c] describes those individuals that stand in the relation r to the individual denoted by c
- [ALL r d] describes those individuals that stand in relation r only to individuals that are described by d
- [AND d_1 ... d_k] describes those individuals that are described by all of the d_i.

For example:

```
[AND  Company
      [EXISTS  7 :Director]
      [ALL  :Manager [AND Woman
                          [FILLS  :Degree phD]]]
      [FILLS  :MinSalary $24.00/hour]]
```

"a company with at least 7 directors, whose managers are all women with PhDs, and whose min salary is $24/hr"

A DL knowledge base

A DL knowledge base is a set of DL sentences serving mainly to

- give names to definitions

  e.g. (FatherOfDaughters $\doteq$ [AND Male [EXISTS 1 :Child] [ALL :Child Female]])

  "A FatherOfDaughters is precisely a male with at least one child and all of whose children are female"

- give names to partial definitions

  e.g. (Dog $\sqsubseteq$ [AND Mammal Pet CarnivorousAnimal [FILLS :VoiceCall barking]])

  gives necessary but not sufficient conditions

  "A dog is among other things a mammal that is a pet and a carnivorous animal whose voice call includes barking"

- assert properties of individuals

  e.g. (joe $\rightarrow$ [AND FatherOfDaughters Surgeon]])

  "Joe is a FatherOfDaughters and a Surgeon"

Other types of DL sentences are typically not used in a KB.

e.g. ([AND Rational Animal] $\doteq$ [AND Featherless Biped])

Formal semantics

Interpretation $\mathfrak{I} = \langle D, I \rangle$ as in FOL, where

- for every constant c, $I[c] \in D$
- for every atomic concept a, $I[a] \subseteq D$
- for every role r, $I[r] \subseteq D \times D$

We then extend the interpretation to all concepts as subsets of the domain as follows:

- I[Thing] = D
- I[[ALL r d]] = {$x \in D$ | for any y, if <x,y> $\in I[r]$ then $y \in I[d]$}
- I[[EXISTS n r]] = {$x \in D$ | there are at least n y such that <x,y> $\in I[r]$}
- I[[FILLS r c]] = {$x \in D$ | <x,$I[c]$> $\in I[r]$}
- I[[AND d_1 ... d_k]] = $I[d_1] \cap ... \cap I[d_k]$

A sentence of DL will then be true or false as follows:

- $\mathfrak{I} \models (d \sqsubseteq e)$ iff $I[d] \subseteq I[e]$
- $\mathfrak{I} \models (d \doteq e)$ iff $I[d] = I[e]$
- $\mathfrak{I} \models (c \rightarrow e)$ iff $I[c] \in I[e]$

Entailment and reasoning

Entailment in DL is defined as in FOL:

A set of DL sentences S entails a sentence α (which we write $S \models \alpha$) iff
for every $\mathfrak{I}$, if $\mathfrak{I} \models S$ then $\mathfrak{I} \models \alpha$

A sentence is valid iff it is entailed by the empty set.

Given a KB consisting of DL sentences, there are two basic sorts of reasoning we consider:

1. determining if KB $\models (c \rightarrow e)$
   whether a named individual satisfies a certain description
2. determining if KB $\models (d \sqsubseteq e)$
   whether one description is subsumed by another

the other case, KB $\models (d \doteq e)$ reduces to
KB $\models (d \sqsubseteq e)$ and KB $\models (d \sqsubseteq e)$

Entailment vs. validity

In some cases, an entailment will hold because the sentence in question is valid.

- ([AND Doctor Female] $\sqsubseteq$ Doctor)
- ([FILLS :Child sue] $\sqsubseteq$ [EXISTS 1 :Child])
- (john $\rightarrow$ [ALL :Hobby Thing])

But in most other cases, the entailment depends on the sentences in the KB.

For example,

([AND Surgeon Female] $\sqsubseteq$ Doctor)

is not valid.

But it is entailed by a KB that contains

(Surgeon $\doteq$ [AND Specialist [FILLS :Specialty surgery]])

(Specialist $\sqsubseteq$ Doctor)

Computing subsumption

We begin with computing subsumption, that is, determining whether or not KB $\models (d \sqsubseteq e)$.

and therefore whether $d \doteq e$

Some simplifications to the KB:

- we can remove the $(c \rightarrow d)$ assertions from the KB
- we can replace $(d \sqsubseteq e)$ in KB by $(d \doteq [\text{AND } e\ a])$, where a is a new atomic concept
- we assume that in the KB for each $(d \doteq e)$, the d is atomic and appears only once on the LHS
- we assume that the definitions in the KB are acyclic

  vs. cyclic $(d \doteq [\text{AND } e\ f])$, $(e \doteq [\text{AND } d\ g])$

Under these assumptions, it is sufficient to do the following:

- normalization: using the definitions in the KB, put d and e into a special normal form, d' and e'
- structure matching: determine if each part of e' is matched by a part of d'.

Normalization

Repeatedly apply the following operations to the two concepts:

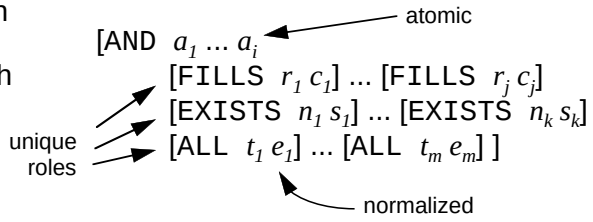
- expand a definition: replace an atomic concept by its KB definition
- flatten an AND concept:

  [AND ... [AND $d\ e\ f$] ...] $\Rightarrow$ [AND ... $d\ e\ f$ ...]
- combine the ALL operations with the same role:

  [AND ... [ALL $r\ d$] ... [ALL $r\ e$] ...] $\Rightarrow$ [AND ... [ALL r [AND $d\ e$]] ...]
- combine the EXISTS operations with the same role:

  [AND ... [EXISTS $n_1\ r$] ... [EXISTS $n_2\ r$] ...] $\Rightarrow$
  [AND ... [EXISTS $n\ r$] ...] (where n =Max(n_1,n_2))
- remove a vacuous concept: Thing, [ALL r Thing], [AND]
- remove a duplicate expression

In the end, we end up with a normalized concept of the following form



Normalization example

[AND Person
[ALL :Friend Doctor]
[EXISTS 1 :Accountant]
[ALL :Accountant [EXISTS 1 :Degree]]
[ALL :Friend Rich]
[ALL :Accountant [AND Lawyer [EXISTS 2 :Degree]]]]

⇩

[AND Person
[EXISTS 1 :Accountant]
[ALL :Friend [AND Rich Doctor]]
[ALL :Accountant [AND Lawyer [EXISTS 1 :Degree] [EXISTS 2 :Degree]]]]

⇩

[AND Person
[EXISTS 1 :Accountant]
[ALL :Friend [AND Rich Doctor]]
[ALL :Accountant [AND Lawyer [EXISTS 2 :Degree]]]]

Structure matching

Once we have replaced atomic concepts by their definitions, we no longer need to use the KB.

To see if a normalized concept [AND e_1 .... e_m] subsumes a normalized concept [AND d_1 ... d_n], we do the following:

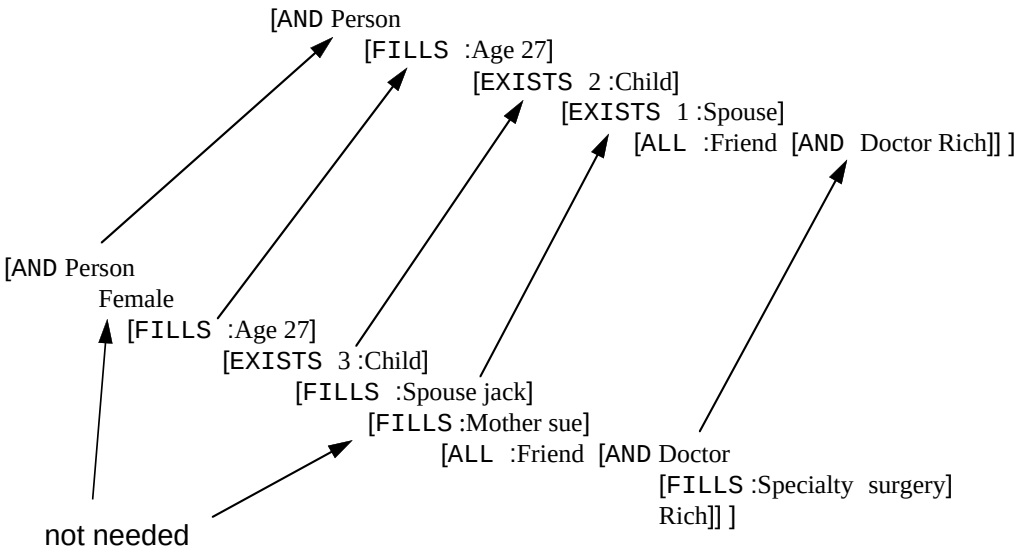
For each component e_j, check that there is a matching component d_i, where

- if e_j is atomic or [FILLS r c], then d_i must be identical to it;
- if e_j = [EXISTS 1 r], then d_i must be [EXISTS n r] or [FILLS r c];
- if e_j = [EXISTS n r] where n >1, then d_i must be of the form [EXISTS m r] where $m \geq n$;
- if e_j = [ALL r e'], then d_i must be [ALL r d'], where recursively e' subsumes d'.

In other words, for every part of the more general concept, there must be a corresponding part in the more specific one.

It can be shown that this procedure is sound and complete: it returns YES iff KB $\models (d \sqsubseteq e)$.

Structure matching example



Computing satisfaction

To determine if KB $\models (c \rightarrow e)$, we use the following procedure:

1. find the most specific concept d such that KB $\models (c \rightarrow d)$
2. determine whether or not KB $\models (d \sqsubseteq e)$, as before.

To a first approximation, the d we need is the AND of every d_i such that $(c \rightarrow d_i) \in$ KB.

However, this can miss some inferences!

Suppose the KB contains

(joe $\rightarrow$ Person)
(canCorp $\rightarrow$ [AND Company
[ALL :Manager Canadian]
[FILLS :Manager joe]]

then the KB $\models$ (joe $\rightarrow$ Canadian).

To find the d, a more complex procedure is used that *propagates* constraints from one individual (canCorp) to another (joe).

The individuals we need to consider need not be named by constants; they can be individuals that arise from EXISTS (like Skolem constants).

Taxonomies

Two common sorts of queries in a DL system:

- given a query concept q, find all constants c such that KB $\models (c \rightarrow q)$

  e.g. q is [AND Stock FallingPrice MyHolding] might want to trigger a procedure for each such c

- given a query constant c, find all *atomic* concepts a such that KB $\models (c \rightarrow a)$

We can exploit the fact that concepts tend to be structured hierarchically to answer queries like these more efficiently.

Taxonomies arise naturally out of a DL KB:

- the nodes are the atomic concepts that appear on the LHS of a sentence $(a \sqsubseteq d)$ or $(a \doteq d)$ in the KB
- there is an edge from a_i to a_j if $(a_i \sqsubseteq a_j)$ is entailed and there is no distinct a_k such that $(a_i \sqsubseteq a_k)$ and $(a_k \sqsubseteq a_j)$.

  can link every constant c to the most specific atomic concepts a in the taxonomy such that KB $\models (c \rightarrow a)$

Positioning a new atom in a taxonomy is called classification

Computing classification

Consider adding ($a_{new} \doteq d$) to the KB.

- find S, the most specific subsumers of d: the atoms a such that KB $\models (d \sqsubseteq a)$, but nothing below a see below
- find G, the most general subsumees of d: the atoms a such that KB $\models (a \sqsubseteq d)$, but nothing above a

  if $S \cap G$ is not empty, then a_{new} is not new
- remove any links from atoms in G to atoms in S
- add links from all the atoms in G to a_{new} and from a_{new} to all the atoms in S
- reorganize the constants:
  for each constant c such that KB $\models (c \rightarrow a)$ for all $a \in S$,
  but KB $\models (c \rightarrow a)$ for no $a \in G$, and where KB $\models (c \rightarrow d)$,
  remove links from c to S and put a single link from c to a_{new}.

Adding ($a_{new} \sqsubseteq d$) is similar, but with no subsumees.

Subsumers and subsumees

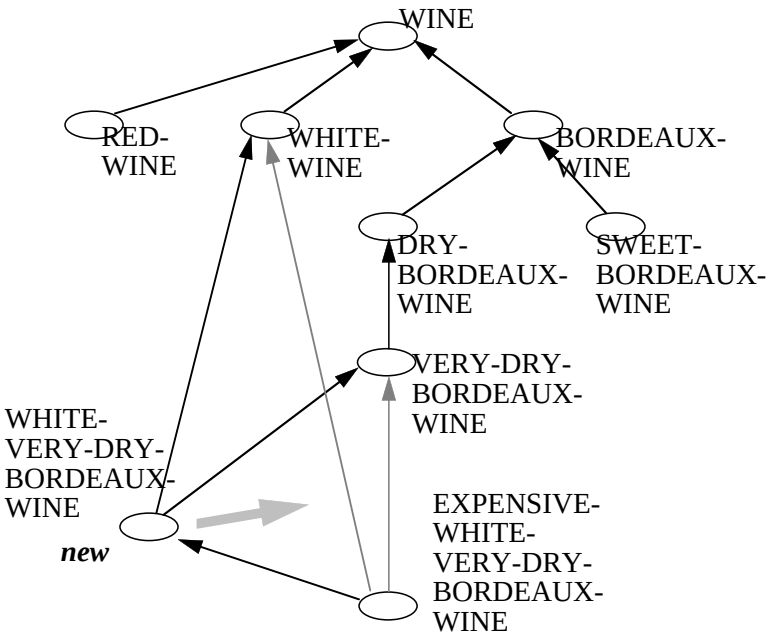
Calculating the most specific subsumers of a concept d:

- Start with $S = \{\text{Thing}\}$.
- Repeatedly do the following:
  - Suppose that some $a \in S$ has at least one child a' just below it in the taxonomy such that KB $\models (d \sqsubseteq a')$.
  - Then remove a from S and replace it by all such children a'.

Calculating the most general subsumees of a concept d:

- Start with G = the most specific subsumers.
- Repeatedly do the following:
  - Suppose that for some $a \in G$, KB $\not\models (a \sqsubseteq d)$.
  - Then remove a from G and replace it by all of its children (or delete it, if there are none).
- Repeatedly delete any element of G that has a parent subsumed by d.

An example of classification



Using the taxonomic structure

Note that classification uses the structure of the taxonomy:

> If there is an a' just below a in the taxonomy such that KB $\not\models (d \sqsubseteq a')$, we never look below this a'. If this concept is sufficiently high in the taxonomy (e.g. just below Thing), an entire subtree will be ignored.

Queries can also exploit the structure:

> For example, to find the constants described by a concept q, we simply classify q and then look for constants in the part of the taxonomy subtended by q. The rest of the taxonomy not below q is ignored.

This natural structure allows us to build and use very large knowledge bases.

- the time taken will grow linearly with the *depth* of the taxonomy
- we would expect the depth of the taxonomy to grow *logarithmically* with the size of the KB
- under these assumptions, we can handle a KB with thousands or even millions of concepts and constants.

Taxonomies vs. frame hierarchies

The taxonomies in DL look like the **IS-A** hierarchies with frames.

There is a big difference, however:

- in frame systems, the KB designer gets to decide what the fillers of the :IS-A slot will be; the :IS-A hierarchy is constructed manually
- in DL, the taxonomy is completely determined by the meaning of the concepts and the subsumption relation over concepts

For example, a concept such as

[AND Fish [FILLS :Size large]]

must appear in the taxonomy below Fish even if it was first constructed to be given the name Whale. It cannot simply be positioned below Mammal.

To correct our mistake, we need to associate the name with a different concept:

[AND Mammal [FILLS :Size large] ...]

Inheritance and propagation

As in frame hierarchies, atomic concepts in DL inherit properties from concepts higher up in the taxonomy.

For example, if a Doctor has a medical degree, and Surgeon is below Doctor, then a Surgeon must have a medical degree.

This follows from the logic of concepts:

If KB $\models$ (Doctor $\sqsubseteq$ [EXISTS 1 :MedicalDegree])
and KB $\models$ (Surgeon $\sqsubseteq$ Doctor)
then KB $\models$ (Surgeon $\sqsubseteq$ [EXISTS 1 :MedicalDegree])

This is a simple form of *strict* inheritance (*cf*. next chapter)

Also, as noted in computing satisfaction (e.g. with joe and canCorp), adding an assertion like ($c \rightarrow e$) to a KB can cause other assertions ($c' \rightarrow e'$) to be entailed for other individuals.

This type of propagation is most interesting in applications where membership in classes is monitored and changes are significant.

Extensions to the language

A number of extensions to the DL language have been considered in the literature:

- upper bounds on the number of fillers

  [AND [EXISTS 2 :Child] [AT-MOST 3 :Child]]

  opens the possibility of inconsistent concepts

- sets of individuals: [ALL :Child [ONE-OF wally theodore]]
- relating the role fillers: [SAME-AS :President :CEO]
- qualified number restriction: [EXISTS 2 :Child Female] vs.
  [AND [EXISTS 2 :Child] [ALL :Child Female]]
- complex (non-atomic) roles: [EXISTS 2 [RESTR :Child Female]]

  [ALL [RESTR :Child Female] Married] vs.
  [ALL :Child [AND Female Married]]

Each of these extensions adds extra complexity to the problem of calculating subsumption.

This topic will be explored for RESTR in Chapter 16.

Some applications

Like production systems, description logics have been used in a number of sorts of applications:

- interface to a DB

  relational DB, but DL can provide a nice higher level view of the data based on objects

- working memory for a production system

  instead of a having rules to reason about a taxonomy and inheritance of properties, this part of the reasoning can come from a DL system

- assertion and classification for monitoring

  incremental change to KB can be monitored with certain atomic concepts declared “critical”

- contradiction detection in configuration

  for a DL that allows contradictory concepts, can alert the user when these are detected. This works well for incremental construction of a concept representing e.g. a configuration of a computer.