CS 2740 Knowledge Representation
Lecture 14

Structured descriptions

Milos Hauskrecht
milos@cs.pitt.edu
5329 Sennott Square

Based on lecture notes by Brachman and Levesque

Atomic predicates and description logic

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

However, predicates have an internal structure and connections to other predicates.

- e.g. A married person vs person predicates
- E.g. A coffee table vs table predicates

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

- In this lecture we will examine a logic that allows us to have both **atomic and non-atomic predicates**: a description logic

Concepts, roles, constants

Description logic: sentences are either true or false

Three sorts of expressions:

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

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

Description logic:

- concepts need not be atomic and can have semantic relationships to each other: e.g. Student vs GraduateStudent
- roles will remain atomic

Description logic: syntax

- Three types of non-logical symbols:
  - **atomic concepts**: Dog, Teenager, GraduateStudent
    we also 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**: $\rightarrow$, $\doteq$, $\sqsubseteq$

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 *d1*, ..., *dk* are concepts, then so is [AND *d1*, ..., *dk*].
- Three **types of sentences** in DL:
  - If *d* and *e* are concepts, then ($d \doteq e$) is a sentence.
  - if *d* and *e* are concepts, then ($d \sqsubseteq e$) is a sentence.
  - If *d* is a concept and *c* is a constant, then ($c \rightarrow d$) is a sentence.

Syntax of DL

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

- [EXISTS *n r*] individuals that stand in relation *r* to at least *n* other individuals
- [FILLS *r c*] individuals that stand in the relation *r* to the individual denoted by *c*
- [ALL *r d*] individuals that stand in relation *r* only to individuals that are described by *d*
- [AND *d1* ... *dk*] individuals that are described by all of the *di*.

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 that

- give **names to definitions (defines)**

  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 (subsumes)**

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

  "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 (satisfies)**

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

  "Joe is a FatherOfDaughters and a Surgeon"

Entailment in DL

Entailment in DL is defined as in FOL:

- A set of DL sentences *S* entails a sentence *a* (which we write *S* |= *a*) iff for every interpretation under which S is true , *a* is true as well

- Given a KB consisting of DL sentences, there are two basic sorts of reasoning we consider:
  - determining if KB |= ($c \rightarrow e$)
    whether a named individual satisfies a certain description
  - determining if KB |= ($d \sqsubseteq e$)
    whether one description is subsumed by another
  - the other case, KB |= ($d \doteq e$) reduces to
    KB |= ($d \sqsubseteq e$) and KB |= ($d \rightarrow e$)

Normalization

atomic

[AND *a1* ... *ai*
  [FILLS *r1 c1*] ... [FILLS *rj cj*]
  [EXISTS *n1 s1*] ... [EXISTS *nk sk*]
  [ALL *t1 e1*] ... [ALL *tm em*]]

unique roles

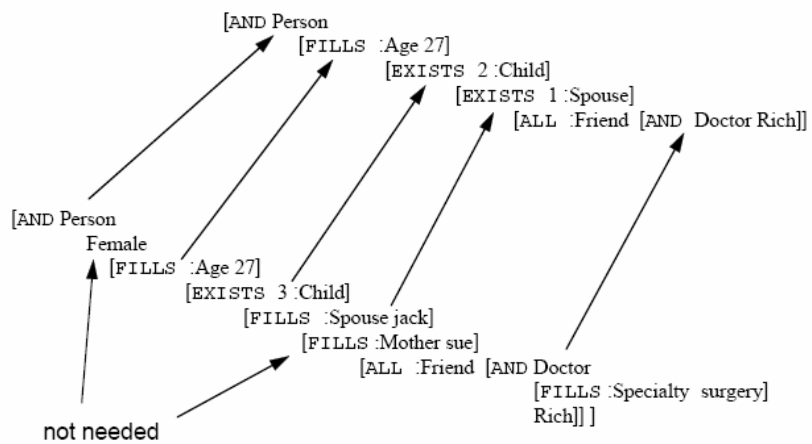
Normalization example

[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 example



Computing satisfaction

To determine if KB |= (*c* → *e*), we use the following two-step procedure:

- find the most specific concept *d* such that KB = (*c* → *d*)
- determine whether or not KB |=(*d* ⊑ *e*), as before.

- To a first approximation, the *d* we need is the AND of every *di* such that (*c* → *di*) ∈ KB
- But suppose the KB contains

  (joe → Person)
  (canCorp → [AND Company
  [ALL :Manager Canadian]
  [FILLS :Manager joe]]
- then the KB |= (joe → 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]
- 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**

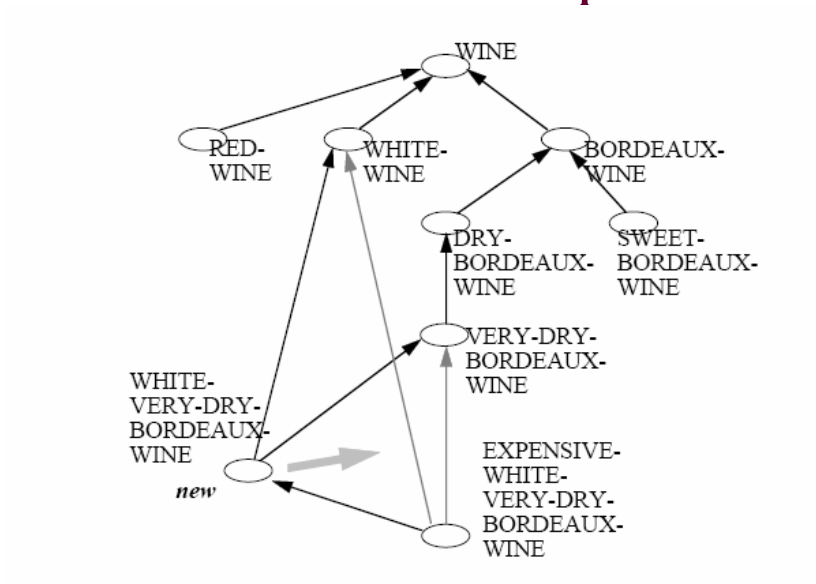
Classification

Consider adding $(a \doteq d)$ 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
- 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 is not new
- remove any links from atoms in G to atoms in S
- add links from all the atoms in G to a and from a 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.

Adding $(a \sqsubseteq d)$ is similar, but with no subsumees.

Classification example



Using 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 in 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 |= (Doctor $\sqsubseteq$ [EXISTS 1 :MedicalDegree])

  and KB |=(Surgeon $\sqsubseteq$ Doctor)

  then KB |=(Surgeon $\sqsubseteq$ [EXISTS 1 :MedicalDegree])

This is a simple form of *strict* inheritance

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

- 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.

Applications

Like production systems, description logics have been used in a number 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.