CS 2740 Knowledge Representation
Lecture 13

Structured descriptions

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

Based on lecture notes by Brachman and Levesque

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.

- 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

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. 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.

Difference:

- unlike in FOL, concepts need not be atomic and can have semantic relationships to each other: e.g. Student 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

- 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 *d1* ... *dk*] describes those 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 serving mainly to

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

  e.g. (FatherOfDaughters ≐
  [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 ⊑ [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 → [AND FatherOfDaughters Surgeon]])

  "Joe is a FatherOfDaughters and a Surgeon"

Semantics of DL

Interpretation similar to the FOL:

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

Extend the interpretation to all concepts as subsets of the domain:

- $I[\text{Thing}] = D$
- $I[[\text{ALL } r\ d]] = \{x \in D \mid \text{for any } y, \text{ if } \langle x,y \rangle \in I[r] \text{ then } y \in I[d]\}$
- $I[[\text{EXISTS } n\ r]] = \{x \in D \mid \text{there are at least } n\ ys \text{ such that } \langle x,y \rangle \in I[r]\}$
- $I[[\text{FILLS } r\ c]] = \{x \in D \mid \langle x, I[c] \rangle \in I[r]\}$
- $I[[\text{AND } d1 \ldots dk]] = I[d1] \cap \ldots \cap I[dk]$

Semantics of DL

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

- **subsumes**
  ($d \sqsubseteq e$) iff $I[d] \subseteq I[e]$
- **defines**
  ($d \doteq e$) iff $I[d] = I[e]$
- **satisfies**
  ($c \rightarrow e$) iff $I[c] \in I[e]$

Entailment in DL

Entailment in DL is defined as in FOL:

- A set of DL sentences S entails a sentence a (which we write $S \models a$) 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 $\models (c \rightarrow e)$
    whether a named individual satisfies a certain description
  – 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 $\models (d \rightarrow e)$

Entailment and validity

In some cases, an entailment will hold because the sentence in question is valid (true for all interpretations).

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

But in 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 |= ($d \sqsubseteq e$).

Some simplifications to the KB:

- we can remove ($c \rightarrow d$) assertions from the KB
- we can replace ($d \sqsubseteq e$) in KB by ($d \doteq$ [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 (example: $d \doteq$ [AND e f]), ($e \doteq$ [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*] ...] → [AND ... *d e f* ...]
- combine the ALL operations with the same role:
  [AND ... [ALL *r d*] ... [ALL *r e*] ...] → [AND ... [ALL *r* [AND *d e*]] ...]
- combine the EXISTS operations with the same role:
  [AND ... [EXISTS *n1 r*] ... [EXISTS *n2 r*] ...] →
  [AND ... [EXISTS *n r*] ...] (where *n* =Max(*n1,n2*))
- remove a vacuous concept: Thing, [ALL *r* Thing], [AND]
- remove a duplicate expression

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

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]]]]

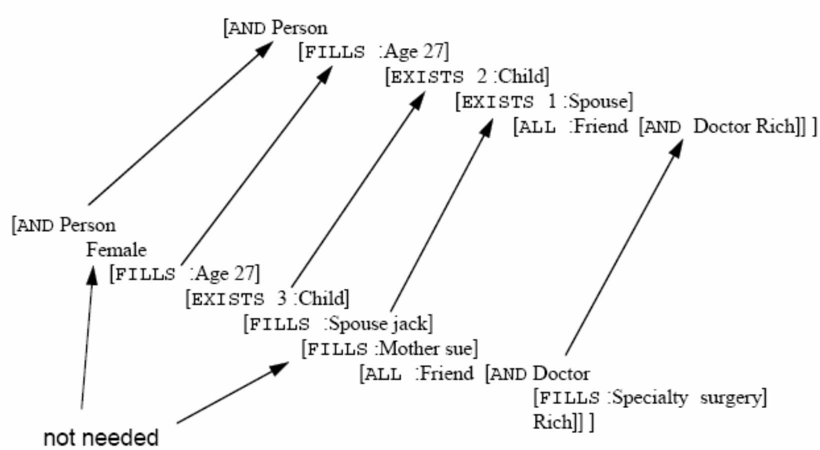
Structure matching

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

To see if a normalized concept [AND *e1* .... *em*] subsumes a normalized concept [AND *d1* ... *dn*], we do the following:

- For each component *ej,* check that there is a matching component *di*, where
  - if *ej* is atomic or [FILLS *r c*], then *di* must be identical to it;
  - if *ej* = [EXISTS 1 *r*], then *di* must be [EXISTS *n r*] or [FILLS *r c*];
  - if *ej* = [EXISTS *n r*] where *n* >1, then *di* must be of the form [EXISTS *m r*] where $m \geq n$;
  - if *ej* = [ALL *r e'*], then *di* 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 |=($d \sqsubseteq e$).

Structure matching example



Computing satisfaction

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

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

- To a first approximation, the d we need is the AND of every di such that ($c \rightarrow di$) $\in$ KB
- 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 |=($c \rightarrow q$)
  e.g. q is [AND Stock FallingPrice MyHolding]
- given a query constant c, find all *atomic* concepts a such that KB |= ($c \rightarrow a$)

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 ai to aj if ($ai \sqsubseteq aj$) is entailed and there is no distinct ak such that ($ai \sqsubseteq ak$) and ($ak \sqsubseteq aj$).
  – can link every constant c to the most specific atomic concepts a in the taxonomy such that KB |=($c \rightarrow a$)

Positioning a new atom in a taxonomy is called **classification**

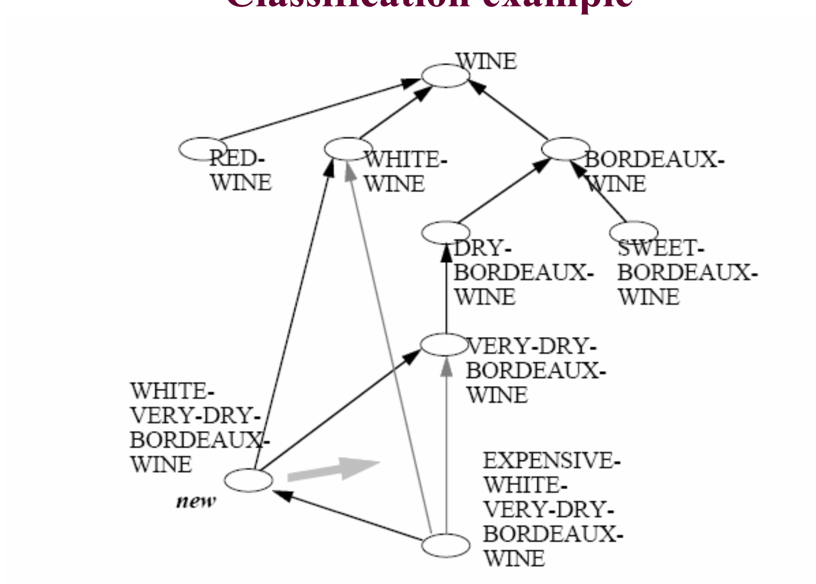
Classification

Consider adding ($a \doteq d$) to the KB.

- find S, the most specific subsumers of d: the atoms a such that KB |= ($d \sqsubseteq a$), but nothing below a
- find G, the most general subsumees of d: the atoms a such that KB |= ($a \sqsubseteq d$), 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 |= ($c \rightarrow a$) for all $a \in S$, but KB |= ($c \rightarrow a$) for no $a \in G$, and where KB |= ($c \rightarrow d$), 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 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.