Description logic

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:** →, ⊤, ⊥
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 \([\text{ALL} \ r \ d]\) is a concept.
  - individuals that stand in relation \( r \) only to individuals that are described by \( d \)
- If \( r \) is a role and \( n \) is an integer, then \([\text{EXISTS} \ n \ r]\) is a concept.
  - individuals that stand in relation \( r \) to at least \( n \) other individuals
- If \( r \) is a role and \( c \) is a constant, then \([\text{FILLS} \ r \ c]\) is a concept.
  - individuals that stand in the relation \( r \) to the individual denoted by \( c \)
- If \( d_1, \ldots, d_k \) are concepts, then so is \([\text{AND} \ d_1, \ldots, d_k]\).
  - individuals that are described by all of the \( d_i \)

Semantic of DL

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

- \([\text{EXISTS} \ n \ r]\) individuals that stand in relation \( r \) to at least \( n \) other individuals
- \([\text{FILLS} \ r \ c]\) individuals that stand in the relation \( r \) to the individual denoted by \( c \)
- \([\text{ALL} \ r \ d]\) individuals that stand in relation \( r \) only to individuals that are described by \( d \)
- \([\text{AND} \ d_1 \ldots \ d_k]\) individuals that are described by all of the \( d_i \).

Example

- \([\text{AND} \ \text{Company} \ [\text{EXISTS} \ 7 \ :\text{Director}] \ [\text{ALL} \ :\text{Manager} \ [\text{AND} \ :\text{Woman} \ [\text{FILLS} \ :\text{Degree} \text{phD}]]) \ [\text{FILLS} \ :\text{MinSalary} \$24.00/\text{hour}]]\)

  “a company with at least 7 directors, whose managers are all women with PhDs, and whose min salary is $24/hr”
Sentences in DL

- Three types of sentences in DL:
  - If \( d \) and \( e \) are concepts, then \((d \sqsubseteq e)\) is a sentence:
    - \( e \) defines \( d \) (or \( d \) is defined by \( e \))
  - If \( d \) and \( e \) are concepts, then \((d \equiv e)\) is a sentence.
    - \( d \) subsumes \( e \) (or \( e \) is subsumed by \( d \))
  - If \( d \) is a concept and \( c \) is a constant, then \((c \to d)\) is a sentence.
    - \( c \) satisfies \( d \)

A DL knowledge base

A DL knowledge base is a set of DL sentences that

- give names to definitions (defines)
  e.g. (FatherOfDaughters \( \sqsubseteq \)
       [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 \( \equiv \) [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 \( \to \) [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 $S \models a$) 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 $KB \models (c \to e)$ 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, $KB \models (d \sqsubseteq e)$ reduces to $KB \models (d \sqsubseteq e)$ and $KB \models (e \sqsubseteq d)$

Normalization

atomic

\[
\text{[AND } a_1 \ldots a_i \text{ ] [FILLS } r_1 c_1 \ldots [FILLS r_j c_j] \text{ ] [EXISTS } n_1 s_1 \ldots [EXISTS n_k s_k] \text{ ] [ALL } t_1 e_1 \ldots [ALL t_m e_m] \text{ ]}
\]
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]]]]

Sub-sumption inference: Structure matching example
Satisfaction: inference

To determine if $\text{KB} \models (c \rightarrow e)$, we use the following two-step procedure:

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

• To a first approximation, the $d$ we need is the AND of every $d_i$ such that $(c \rightarrow d_i) \in \text{KB}$
• But 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 $\text{KB} \models (c \rightarrow q)$
  e.g. $q$ is [AND Stock FallingPrice MyHolding]
• given a query constant $c$, find all atomic concepts $a$ such that
  $\text{KB} \models (c \rightarrow a)$

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 $(a \sqsupset d)$ 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 $\text{KB} \models (c \rightarrow a)$

Positioning a new atom in a taxonomy is called classification.
Classification

Consider adding \((a \sqsupset d)\) to the KB.
- find \(S\), the most specific subsumers of \(d\): the atoms \(a\) such that \(\text{KB} \models (d \sqsubseteq a)\), but nothing below \(a\)
- find \(G\), the most general subsumees of \(d\): the atoms \(a\) such that \(\text{KB} \models (a \sqsubseteq d)\), 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 \(\text{KB} \models (c \rightarrow a)\) for all \(a \in S\), but \(\text{KB} \not\models (c \rightarrow a)\) for no \(a \in G\), and where \(\text{KB} \models (c \rightarrow d)\), 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 \( \models (d \sqsubseteq a') \), 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 $KB \models (\text{Doctor} \in [\exists 1 : \text{MedicalDegree}])$
  and $KB \models (\text{Surgeon} \in \text{Doctor})$
  then $KB \models (\text{Surgeon} \in [\exists 1 : \text{MedicalDegree}])$

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
    • $[\text{AND} \ [\exists 2 : \text{Child}] [\text{AT-MOST} 3 : \text{Child}]]$
      opens the possibility of inconsistent concepts
  - sets of individuals: $[\text{ALL} : \text{Child} [\text{ONE-OF} \ wally \ theodore]]$
  - relating the role fillers: $[\text{SAME-AS} : \text{President} : \text{CEO}]$
  - qualified number restriction:
    $[\exists \text{EXISTS} 2 : \text{Child Female}]$ vs.
    $[\text{AND} [\exists \text{EXISTS} 2 : \text{Child}] [\text{ALL} : \text{Child Female}]]$
  - complex (non-atomic) roles: $[\exists \text{EXISTS} 2 [\text{RESTR} : \text{Child Female}]]$
    $[\text{ALL} [\text{RESTR} : \text{Child Female} \text{ Married}]]$ vs.
    $[\text{ALL} : \text{Child} [\text{AND} \text{ Female Married}]]$
• 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 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.