Non-monotonic Reasoning

- Motivation
- Non-monotonic reasoning formalisms
- Closed world reasoning (closed world assumption and predicate completion)
- Relational databases in FOL as an example

Example

Imagine the course schedule of a university department available on the Web. How would you represent all relevant information about **who teaches what course** in FOL?

You might have something like:

$$Teaches(Alex, CS100),\ Teaches(Bob, P100),$$

$$Teaches(Charlie, P200)$$

Now answer the following question:

- Who is teaching CS100?

The answer to this question is "Alex" as we can see from the above KB.

Example (cont'd)

Now let us answer the following questions:

- Is Bob teaching CS100?
- Is Alex teaching CS200?

Assuming that the schedule is complete, the answer to both of these questions is "no" but this is not explicit in the schedule KB!

Here we have a situation where **in the absence of information to the contrary**, we assume that Bob is not teaching CS100 and Alex is not teaching CS200.

Example (cont'd)

Now assume that we have just learned that Alex teaches the course CS200 as well.

The KB should now become:

$$Teaches(Alex, CS100),\ Teaches(Bob, P100),$$

$$Teaches(Charlie, P200),\ Teaches(Alex, CS200)$$

Now answer the following question:

- Is Alex teaching CS200?

The answer to this question now is "yes" and it is **different** than the answer we got previously.

Default or Non-monotonic Reasoning

In the previous example we **made an assumption in the absence of information to the contrary, and revised this assumption later** when new knowledge became available.

This is called **default** or **non-monotonic reasoning** and cannot be formalized directly in pure FOL in general.

Monotonicity of FOL

Theorem. Let KB be a set of FOL formulas and α, β two arbitrary FOL formulas. If $KB \models \alpha$ then $KB \cup \{\beta\} \models \alpha$.

The above theorem captures the **monotonicity property** of FOL.

This property does **not** help us in the previous example!

Another Example

Let us now try to represent the following information in FOL:

"Violins have four strings"

How do we represent this information in FOL?

The sentence

$$(\forall x)(Violin(x) \Rightarrow NoOfStrings(x, 4))$$

is not entirely appropriate for our example.

The sentence talks about **all** violins while our sentence is about **prototypical** violins.

In other words, the above universally quantified sentence is only an **approximation** of reality for the world of violins.

Example (cont'd)

But let us assume that we will use this universally quantified sentence to represent the given information.

Now consider the following question:

- How many strings does John's violin have?

The answer to this question is "four" since this is what the above KB gives us.

Example (cont'd)

Now assume that we have just learned that John's violin has one string removed.

We should now be able to **update** the previous KB and in this way **revise our beliefs** about John's violin.

As a result, the answer to the previous question should become "three" and it is different than the answer we got previously.

This kind of reasoning is again **non-monotonic**.

Example (cont'd)

So how do we update the KB?

The sentence

$$(\forall x)(Violin(x) \Rightarrow NoOfStrings(x, 4))$$

could be modified to become

$$(\forall x)(Violin(x) \wedge x \neq ViolinOfJohn \Rightarrow NoOfStrings(x, 4)).$$

If we adopt this representation, we need to **write down an exception for every atypical violin**.

This is usually called the **qualification problem** in the relevant literature (since we need to add a number of qualifications to each rule).

Sources of Default Information

General Statements

- **normal:** Under typical circumstances, Ps are Qs.

  Examples: People work close to where they live. Children enjoy singing.

- **prototypical:** The prototypical P is a Q.

  Examples: Apples are red. Owls hunt at night.

- **statistical:** Most Ps are Qs.

  Example: The people in the waiting room are growing impatient.

Sources of Default Information (cont'd)

Lack of Information to the Contrary

- **familiarity:** If a P was not a Q, you would know it.

  Example: No nation has a political leader more than 2.10 metres tall.

- **group confidence:** All the known Ps are known (or assumed) to be Qs.

  Example: Natural languages are easy for children to learn.

Sources of Default Information (cont'd)

Conventional Uses

- **conversational:** A P is a Q, unless I tell you otherwise.

  Example: Being told "The closest gas station is two blocks east", the assumed default is that the gas station is open.

- **representational:** A P is a Q, unless otherwise indicated.

  Examples: The speed limit in a city. An open door to an office, meaning that the occupant can be disturbed.

Sources of Default Information (cont'd)

Persistence

- **inertia:** A P is a Q unless something changes it.

  Examples: Marital status. The position of objects (within limits).

- **time:** A P is a Q if it used to be a Q.

  Examples: The color of objects. Their sizes.

Non-monotonic Logics

Non-monotonic reasoning has been studied in detail in AI and various **non-monotonic logics** have been invented. Some well-known approaches to non-monotonic reasoning are:

- Closed world reasoning (e.g., closed world assumption or predicate completion).
- Circumscription
- Default logic
- Auto-epistemic logic

We will cover only **closed world reasoning** in this course.

Closed World Reasoning

Let us consider representing information about a world in FOL using a vocabulary of constant, function and predicate symbols.

Typically, only a very small percentage of the large number of atomic sentences that can be formed will be true. A reasonable representation convention is then the following:

- Give explicitly the sentences that are true.
- Assume the unmentioned atomic sentences false.

The Closed World Assumption

The **closed world assumption (CWA)**, originally proposed by Ray Reiter in 1978, is the following:

> Let KB be a knowledge base and ϕ an atomic sentence. If $KB \not\models \phi$ then assume ϕ to be false.

CWA is a non-monotonic reasoning formalism.

The CWA More Formally

Let KB be a knowledge base (i.e., a set of FOL formulas).

Let

$$KB^+ = KB \cup \{\neg\psi : \ \psi \text{ is an atomic sentence and } KB \not\models \psi\}.$$

Then, reasoning under the CWA can be done using a **new entailment relation** $\models_c$ which is defined as follows:

$$KB \models_c \phi \text{ iff } KB^+ \models \phi$$

Exercise

Apply the CWA to the course schedule KB below:

$$Teaches(Alex, CS100),\ Teaches(Bob, P100),$$

$$Teaches(Charlie, P200),\ Teaches(Alex, CS200)$$

Problems with the CWA

The CWA can result in inconsistencies. This depends critically on **syntactic features** e.g., what formulas we have in the KB.

Example: Let KB be

$$Prof(John) \vee Prof(Mary).$$

Then

$$KB^{+} = \{Prof(John) \vee Prof(Mary),\ \neg Prof(John),\ \neg Prof(Mary)\}$$

which is inconsistent!

There are extensions of CWA that deal correctly with arbitrary disjunctions.

Closed World Reasoning in Relational Databases

Relational databases are essentially **simple knowledge bases**. Here we introduce them as a nice example of applying closed word reasoning concepts such as the CWA presented earlier.

We will discuss how to formalize the well-known concepts of relational databases in FOL. This formalization will be **a good example of the power of FOL**, and we will also get nice and simple examples for closed world reasoning.

We will introduce some more interesting KR concepts that we have seen in other parts of the course as well.

Example

TEACHER

NAME
Alex
Bob
Charlie

COURSE

NUMBER
CS100
CS200
P100
P200

STUDENT

NAME
John
Mary
Pam
Paul

Example (cont'd)

TEACHES

NAME	NUMBER
Alex	CS100
Alex	CS200
Bob	P100
Charlie	P200

ENROLLED

NAME	NUMBER
John	CS100
John	P100
Mary	CS100
Pam	P100
Paul	CS200
Paul	P200

FOL and Relational Databases

How can we use concepts of FOL to understand the theory of relational databases?

Two perspectives have been developed in the literature: the **model-theoretic** and the **proof-theoretic** one.

The FO language Corresponding to a Database

In both perspectives, for a given database DB, we define a **FO language** $\mathcal{L}_{DB}$ as follows:

- For each relation R in DB, we have a corresponding predicate **symbol** P_R of the same arity in $\mathcal{L}_{DB}$.
- For each attribute value v in a relation of DB, we have a corresponding **constant** C_v in $\mathcal{L}_{DB}$.
- $\mathcal{L}_{DB}$ has no function symbols.

Notation: To avoid complex notation, in the examples we simply use R and v for the predicates and constants.

The Model-Theoretic Perspective

The given database DB is considered to be an **interpretation** I_{DB} of $\mathcal{L}_{DB}$ with the following properties:

- The universe of the interpretation is the set of all values in the database.
- Each constant C_v is mapped to attribute value v.
- The interpretation of each predicate P_R is given by the relation R.

Queries and Integrity Constraints

The language $\mathcal{L}_{DB}$ can be used to write queries and integrity constraints.

Queries:

$$x : \ Teacher(x) \wedge Teaches(x, CS100)$$

$$: \ Teaches(Charlie, CS100)$$

Integrity Constraints:

$$(\forall x)(Course(x) \Rightarrow (\exists y)(Teacher(y) \wedge Teaches(y, x)))$$

$$(\forall x)(Course(x) \Rightarrow (\exists y)(Student(y) \wedge Enrolled(y, x)))$$

Queries and Integrity Constraints (cont'd)

Answering a query q is equivalent to determining whether the interpretation I_{DB} **satisfies** q.

Verifying that an integrity constraint C **holds** is equivalent to determining whether the interpretation I_{DB} **satisfies** C.

We will not cover the model-theoretic perspective in more detail.

The Proof-Theoretic Perspective

Let DB be a given database. As in the model-theoretic perspective, we can define the FO language $\mathcal{L}_{DB}$.

We can now write a **FO theory (i.e., a set of FO sentences)** T_{DB} that corresponds to DB.

Example (for the above database)

$$Teacher(Alex),\ Teacher(Bob),\ Teacher(Charlie),$$

$$Course(CS100),\ Course(CS200),\ Course(P100),\ Course(P200),$$

$$Student(John),\ Student(Mary),\ Student(Pam),\ Student(Paul),$$

$$Teaches(Alex,\ CS100),\ Teaches(Alex, CS200),$$

$$Teaches(Bob, P100),\ Teaches(Charlie, P200),$$

$$Enrolled(John, CS100),\ Enrolled(John, P100),\ Enrolled(Mary, CS100),$$

$$Enrolled(Pam, P100),\ Enrolled(Paul, CS200),\ Enrolled(Paul, P200)$$

Queries and Integrity Constraints

In the proof-theoretic perspective, the language $\mathcal{L}_{DB}$ can again be used to write queries and integrity constraints.

Queries:

$$x : \ Teacher(x) \wedge Teaches(x, CS100)$$

$$: \ Teaches(Charlie, CS100)$$

Integrity Constraints:

$$(\forall x)(Course(x) \Rightarrow (\exists y)(Teacher(y) \wedge Teaches(y, x)))$$

$$(\forall x)(Course(x) \Rightarrow (\exists y)(Student(y) \wedge Enrolled(y, x)))$$

Queries and Integrity Constraints

But now **answering a query** q is done by determining whether $T_{DB} \models q$ i.e., whether q **logically follows** from T_{DB}. This can be verified using a proof technique e.g., **resolution**.

The same is done for checking that an integrity constraint holds.

Let us try it!

Example

Database:

$$Teacher(Alex),\ Teacher(Bob),\ Teacher(Charlie)$$

$$Course(CS100),\ Course(CS200),\ Course(P100),\ Course(P200)$$

$$Teaches(Alex,\ CS100),\ Teaches(Alex, CS200)$$

$$Teaches(Bob, P100),\ Teaches(Charlie, P200)$$

Queries:

$$:\ Teacher(Alex)$$

$$:\ (\exists x) Course(x)$$

$$:\ (\exists x, y)(Teacher(x) \wedge Course(y) \wedge Teaches(x, y))$$

We can use resolution to see that the answer to all of these queries is "yes".

Example (cont'd)

Database:

$$Teacher(Alex),\ Teacher(Bob),\ Teacher(Charlie)$$

$$Course(CS100),\ Course(CS200),\ Course(P100),\ Course(P200)$$

$$Teaches(Alex,\ CS100),\ Teaches(Alex, CS200)$$

$$Teaches(Bob, P100),\ Teaches(Charlie, P200)$$

Queries:

$$:\ Teacher(CS100)$$

$$:\ \neg Teacher(CS100)$$

The answers to these queries are "no" and "yes" respectively. But resolution will not help us in this case (try it!). How can we solve this problem?

First Solution: CWA

Relational databases silently make the CWA.

If we assume that the above queries are evaluated over T_{DB}^{+} and define query answering using $\models_c$, then the above queries have the answers we would expect.

Other Queries

Can we handle queries such as the following using the CWA?

$$: \ (\forall x)(Teacher(x) \vee Course(x))$$

$$: \ \neg(\exists y)Teaches(y, CS999)$$

Note that both queries essentially involve **universal quantifiers**.

What is the Problem?

We are not able to get the answer "yes" for both of the above queries because T_{DB}^{+} **does not entail** the relevant formulas.

For the query

$$: \ (\forall x)(Teacher(x) \vee Course(x))$$

we can easily find a model of T_{DB}^{+} that includes some domain elements that are not teachers or courses.

For the query

$$: \ \neg(\exists y)Teaches(y, CS999)$$

we can easily find a model of T_{DB}^{+} that includes a domain element who teaches CS999.

The Domain Closure Axiom

If we can express in the knowledge base that **the only individuals are the ones mentioned explicitly in it**, then queries with universal quantifiers can be answered as expected.

This is done by the following **domain closure axiom (DCA)**:

$$(\forall x)(x = C_1 \vee x = C_2 \vee \cdots \vee x = C_n)$$

where $C_1, \ldots, C_n$ are all the constants in the KB.

Example (cont'd)

In the database from the earlier example, the DCA is:

$$(\forall x)(x = Alex \vee x = Bob \vee \cdots \vee x = P100 \vee x = P200)$$

Equality Axioms

Since DCA uses equality, we also need the following axioms for equality:

- **Reflexivity**

$$(\forall x)(x = x)$$

- **Commutativity**

$$(\forall x, y)(x = y \Rightarrow y = x)$$

- **Transitivity**

$$(\forall x, y, z)(x = y \wedge y = z \Rightarrow x = z)$$

- **Substitution of equal terms**

$$(\forall x_1) \cdots (\forall x_n)(\forall y_1) \cdots (\forall y_n)$$

$$(x_1 = y_1 \wedge \cdots x_n = y_n \wedge P(x_1, \ldots, x_n) \Rightarrow P(y_1, \ldots, y_n))$$

Example (cont'd)

If we now start from

- The knowledge base T_{DB}^{+} we constructed using CWA
- The DCA
- The above equality axioms

we can answer the above queries using entailment and resolution (try it!).

The Unique Names Assumption

If we want to allow queries with equality as well, then we also need to make the **unique names assumption (UNA)**.

The UNA states that **distinct names refer to distinct objects.** This is encoded by

$$C_i \neq C_j$$

for all pairs of constants C_i and C_j.

In relational databases (and many knowledge bases), this is a natural assumption to make.

Example

Let the KB be:

$$Teaches(Alex, CS100),\ Teaches(Bob, P100)$$

Then the UNA for this KB is:

$$Alex \neq Bob,\ CS100 \neq P100,$$

$$CS100 \neq Bob,\ CS100 \neq Alex,$$

$$P100 \neq Bob,\ P100 \neq Alex$$

Predicate Completion

There are various ways of expressing in a single sentence of FOL that the only objects that satisfy a predicate are those that **must** do so given what we have stated explicitly (e.g., there are no other teachers expect the ones stated etc.).

The simplest of the relevant techniques is **predicate completion**.

Predicate completion is another closed world reasoning technique that can be used as an alternative to CWA.

Predicate Completion (cont'd)

Let us consider the following simple KB:

$$Teacher(Alex)$$

This KB can be written equivalently as:

$$(\forall x)(x = Alex \Rightarrow Teacher(x))$$

The above formula can be taken as the **"if" part** of the definition for predicate $Teacher$.

Predicate Completion (cont'd)

The assumption that there are no other teachers can now be captured by writing the **"only if"** part of the definition:

$$(\forall x)(Teacher(x) \Rightarrow x = Alex)$$

We can combine both of the above formulas and write:

$$(\forall x)(x = Alex \Leftrightarrow Teacher(x))$$

Predicate Completion (cont'd)

If our KB was

$$Teacher(Alex),\ Teacher(Bob)$$

then the "if" and "only if" forms can be combined as follows:

$$(\forall x)((x = Alex \vee x = Bob) \Leftrightarrow Teacher(x))$$

This is called the **completed definition** of predicate $Teacher$.

Example

Database:

$$Teacher(Alex),\ Teacher(Bob),\ Teacher(Charlie)$$

$$Course(CS100),\ Course(CS200),\ Course(P100),\ Course(P200)$$

$$Teaches(Alex,\ CS100),\ Teaches(Alex, CS200)$$

$$Teaches(Bob, P100),\ Teaches(Charlie, C200)$$

Example (cont'd)

Completion:

$$(\forall x)(Teacher(x) \Leftrightarrow (x = Alex \vee x = Bob \vee x = Charlie))$$

$$(\forall x)(Course(x) \Leftrightarrow (x = CS100 \vee x = CS200 \vee x = P100 \vee x = P200))$$

$$(\forall x)(\forall y)(Teaches(x, y) \Leftrightarrow ((x = Alex \wedge y = CS100) \vee$$

$$(x = Alex \wedge y = CS200) \vee (x = Bob \wedge y = P100) \vee (x = Bob \wedge x = P200)))$$

Example (cont'd)

We can now use resolution on the above formalization **which includes the completion of the KB, the UNA, the DCA and the equality axioms** to answer any of the queries we mentioned earlier:

$$: \ (\exists x)(Teacher(x) \wedge Teaches(x, CS200))$$

$$: \ Teacher(CS100)$$

$$: \ \neg Teacher(CS100)$$

$$: \ (\forall x)(Teacher(x) \vee Course(x))$$

$$: \ \neg(\exists y)Teaches(y, CS999)$$

Predicate Completion for Horn KBs

To apply predicate completion to Horn KBs, the process of producing the completed definitions of predicates is a little more complicated.

The steps of this process are given below.

Predicate Completion for Horn KBs (cont'd)

We can write each Horn clause as

$$(\forall \mathbf{y})(Q_1 \wedge \cdots \wedge Q_m \Rightarrow P(\mathbf{t}))$$

where $\mathbf{t}$ is an n-tuple $(t_1, \ldots, t_n)$ of terms.

There may be no Q_i, in which case the clause is just $P(\mathbf{t})$. The Q_i and $\mathbf{t}$ may contain variables, let us say the tuple of variables $\mathbf{y}$.

Predicate Completion for Horn KBs (cont'd)

The above formula is equivalent to

$$(\forall \mathbf{x})(\forall \mathbf{y})(\mathbf{x} = \mathbf{t} \wedge Q_1 \wedge \cdots \wedge Q_m \Rightarrow P(\mathbf{x}))$$

where $\mathbf{x}$ is a tuple of new variables not occurring in $\mathbf{t}$ or $\mathbf{y}$, and $\mathbf{x} = \mathbf{t}$ is an abbreviation for the conjunction

$$x_1 = t_1 \wedge \cdots \wedge x_n = t_n.$$

Predicate Completion for Horn KBs (cont'd)

Since the variables $\mathbf{y}$ now occur only in the antecedent of the implication, the above is equivalent to:

$$(\forall \mathbf{x})(\exists \mathbf{y})(\mathbf{x} = \mathbf{t} \wedge Q_1 \wedge \cdots \wedge Q_m \Rightarrow P(\mathbf{x}))$$

Predicate Completion for Horn KBs (cont'd)

Let us suppose that we have exactly k clauses for P in our KB. Then we will transform these clauses as above to arrive at:

$$(\forall \mathbf{x})(E_1 \Rightarrow P(\mathbf{x}))$$

$$(\forall \mathbf{x})(E_2 \Rightarrow P(\mathbf{x}))$$

$$\ldots$$

$$(\forall \mathbf{x})(E_n \Rightarrow P(\mathbf{x}))$$

or equivalently

$$(\forall \mathbf{x})(E_1 \vee E_2 \ldots \vee E_n \Rightarrow P(\mathbf{x}))$$

This is the "if" part of the definition of P.

Predicate Completion for Horn KBs (cont'd)

The "only if" completion of P then is:

$$(\forall \mathbf{x})(P(\mathbf{x}) \Rightarrow E_1 \vee E_2 \ldots \vee E_n)$$

The **completed definition** of predicate P is:

$$(\forall \mathbf{x})(E_1 \vee E_2 \ldots \vee E_n \Leftrightarrow P(\mathbf{x}))$$

Example

$$(\forall v)(Ostrich(v) \Rightarrow Bird(v))$$

$$Bird(Tweety)$$

$$Ostrich(Sam)$$

The above knowledge base KB represents the following information:

All ostriches are birds. Tweety is a bird. Sam is an ostrich.

Example (cont'd)

The completed definitions for predicates $Bird$ and $Ostrich$ are:

$$(\forall x)((Ostrich(x) \vee x = Tweety) \Leftrightarrow Bird(x))$$

$$(\forall x)(x = Sam \Leftrightarrow Ostrich(x))$$

Negation as Failure

Predicate completion provides the theoretical basis for the semantics of **negation-as-failure** in logic programming languages (e.g., Prolog) proposed by Keith Clarke in 1978.

See the book by Lloyd in the readings for more details.

Query Answering in Horn KBs

We can start from the completed definitions of predicates, the UNA, the DCA and the equality axioms, and use resolution to answer queries such as the ones presented above for the student-course database.

There are special resolution methods for Horn KBs (SLD and SLDNF resolution) that are more efficient than general resolution. See the book by Lloyd in the readings for more details.

Readings

- Stuart Russell and Peter Norvig. Artificial Intelligence: A Modern Approach, Prentice Hall, 2nd edition (2002).
  `www.cs.berkeley.edu/~russell/aima.html`.
  Chapter 10.
- Ronald J. Brachman and Hector J. Levesque. Knowledge Representation and Reasoning. Morgan Kaufmann, 2004.
  Chapter 11.
- J. W. Lloyd. Foundations of Logic Programming. Springer, 1984.
  Chapter 3.

Readings (cont'd)

- Ray Reiter. Towards a Logical Reconstruction of Relational Database Theory. In M. L. Brodie, J. Mylopoulos and J. W. Schmidt (eds.) On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases and Programming Languages. Springer-Verlag, 1984.