491 Knowledge Representation

Logic Databases (Knowledge Bases)

Marek Sergot
Department of Computing
Imperial College, London

January 2011 v1.0e

Notation

$\mathcal{L}$ is some (logical) language, usually propositional or (a fragment of) first-order predicate logic. Unless stated otherwise, $\mathcal{L}$ is closed under truth-functional operations. (Thus, if $\alpha \in \mathcal{L}$ then $\neg\alpha \in \mathcal{L}$, and if $\alpha \in \mathcal{L}$ and $\beta \in \mathcal{L}$ then $\alpha \vee \beta \in \mathcal{L}$, $\alpha \wedge \beta \in \mathcal{L}$, $\alpha \rightarrow \beta \in \mathcal{L}$, etc.)
Lower-case Greek letters $\alpha, \beta, \ldots$ range over formulas, lower-case Latin letters, $p, q, r, \ldots$ represent atomic formulas.
Upper-case letters $A, B, C, \ldots, X, Y, \ldots$ represent sets of formulas.
(Sometimes I say 'sentence' instead of 'formula'.)

$\mathcal{M}$ is a model for $\mathcal{L}$ (i.e., an interpretation of formulas of $\mathcal{L}$).

$\mathcal{M} \models \alpha$ means that formula α evaluates to true in model $\mathcal{M}$; $\mathcal{M}$ is a model of α.
$\models \alpha$ means that formula α evaluates to true in all models for $\mathcal{L}$.
$A \models \alpha$ means that α is true in all models of A, i.e. $\models (\bigwedge A \rightarrow \alpha)$ when A is finite.

($\bigwedge A$ denotes the conjunction of all formulas of A.)
For convenience (laziness) I will sometimes write $A \models B$ when B is a set of formulas.
$A \models B$ means $A \models \beta$ for all $\beta \in B$, i.e. $A \models \bigwedge B$ when B is finite.
$\mathrm{Th}(A)$ stands for the set of all classical truth-functional consequences of A, i.e.

$$\mathrm{Th}(A) \stackrel{\text{def}}{=} \{\alpha \in \mathcal{L} \mid A \models \alpha\}$$

$\mathrm{Cn}(A)$ is the set of all (not necessarily classical) consequences of A. (We will look at various examples of Cn).
$A \vdash \alpha$ means the same as $\alpha \in \mathrm{Cn}(A)$.
For convenience (laziness): When B is a set of formulas, $A \vdash B$ means $A \vdash \beta$ for all $\beta \in B$, i.e., $B \subseteq \mathrm{Cn}(A)$.

Logic databases/knowledge bases

Two main ways of applying logic to databases/knowledge bases:

- database as *model*
- database as *theory*

(Some databases can be viewed either way, as explained later.)

Database/knowledge base as model

We have a language $\mathcal{L}$ in which to express queries (and perhaps integrity constraints).
The database is a structure $\mathcal{M}$ which can be used to evaluate formulas of $\mathcal{L}$.
$\mathcal{M}$ is a model for $\mathcal{L}$ (i.e., an interpretation of formulas of $\mathcal{L}$).
(Sometimes another formalism/language $\mathcal{L}'$ is used to define the model $\mathcal{M}$.)

A query is $\underline{x} : \alpha(\underline{x})$?
$\alpha(\underline{x})$ is any formula of $\mathcal{L}$ whose free variables are $\underline{x}$.
A query containing no free variables is a 'closed query'.

Answers to query $\underline{x} : \alpha(\underline{x})$? on database $\mathcal{M}$ are those 'vectors' of constants $\underline{c}$ (or more generally ground terms if the language $\mathcal{L}$ contains function symbols) such that

$$\mathcal{M} \models \alpha(\underline{c})$$

For a closed query α (no variables) the answers are:

Yes if $\mathcal{M} \models \alpha$
No if $\mathcal{M} \models \neg\alpha$

For a two-valued logic there are no 'Don't know' answers (though there are examples of three-valued, four-valued, ... models where this might not be the case; we won't be looking at any such in this course).

Example (very simple)

$\mathcal{L}$ is a propositional language with propositional variables (atomic formulas) p and q.

A possible database is

$$\mathcal{M} = \{p \mapsto \mathsf{t}, q \mapsto \mathsf{f}\}$$

The queries p ? and q ? get answers 'Yes' and 'No' respectively because

$$\mathcal{M} \models p$$
$$\mathcal{M} \models \neg q$$

Example (relational database)

Models for first-order logic consist of a set of relations (together with an interpretation function which says what individual each constant (term) of the language denotes).
Consider the following relational database R containing relations `has-office-in`, `city`, `capital-city`, `company`:

`has-office-in`	`city`
(IBM, Winchester)	Winchester
(IBM, London)	London
(IBM, Paris)	Paris
(MBI, London)	NewYork

`capital-city`	`company`
London	IBM
Paris	MBI

For the query language $\mathcal{L}$ let's take the first-order language with predicates `has-office-in`, `city`, `capital-city`, `company` and constants `IBM`, `MBI`, `Winchester`, `London`, `Paris`, `NewYork`.

To construct a model $\mathcal{M}$, let's take the Herbrand interpretation (constants and predicates denote themselves) and the set of relations R to give the interpretation of the predicates.

Q1. x : `has-office-in(IBM,`x`)`?
has answers
$x =$ `Winchester`
$x =$ `London`
$x =$ `Paris`
Why? Because $\mathcal{M} \models$ `has-office-in(IBM,`x`)` for
$x =$ `Winchester`
$x =$ `London`
$x =$ `Paris`.

Q2. $\exists x$ `has-office-in(IBM,`x`)` ?
gets answer 'Yes' because $\mathcal{M} \models \exists x$ `has-office-in(IBM,`x`)`.

Notice that Q2 is a closed ('yes/no') query. Although it contains a variable x, this variable is not 'free': it is existentially quantified.

Q3. $\exists x$ (`has-office-in(IBM,`x`)` $\wedge$ `capital-city(`x`)`) ?
gets answer 'Yes' because $\mathcal{M} \models \exists x$ (`has-office-in(IBM,`x`)` $\wedge$ `capital-city(`x`)`).

Q4. x : `has-office-in(`x`,NewYork)` ?
gets answer 'No solution' because there is no value of x such that $\mathcal{M} \models$ `has-office-in(`x`,NewYork)`.

Q5. $\exists x$ `has-office-in(`x`,NewYork)` ?
gets answer 'No' because $\mathcal{M} \models \neg\exists x$ `has-office-in(`x`,NewYork)`.

Example: the action language $\mathcal{C}+$

The action language $\mathcal{C}+$ is a formalism for specifying the effects of actions and dealing with the default persistence of facts over time. We will look at $\mathcal{C}+$ in the temporal reasoning section later in the course.

An action description in $\mathcal{C}+$ is a set of 'causal laws' that define a *labelled transition system* of a certain kind. The labelled transition system so defined can be used to evaluate a variety of temporal query languages.

So here we have an example of 'database/knowledge base as model'. An action description in $\mathcal{C}+$ defines a (symbolic representation of) a transition system $\mathcal{M}$. Properties of the transition system are queried/explored by writing a formula α of a language $\mathcal{L}$ (usually a temporal logic of some kind) and then evaluating whether $\mathcal{M} \models \alpha$.

More about this later in the course.

Database/knowledge base as theory

A database or knowledge base is a set D of assumptions (beliefs, data) expressed in language $\mathcal{L}$. Details of the language $\mathcal{L}$ don't matter for now. We will look at specific examples in due course.

The content of the database is $\mathrm{Cn}(D)$.

In logic, $\mathrm{Cn}(D)$ is called a 'theory' (sometimes a 'belief set'). D is the 'base'.

If you want a concrete example, think of D as some set of first-order formulas and Cn as Th. The content of the database is then whatever we can derive in first-order logic from the formulas D.

Or to take another example: suppose D is a logic program (a Prolog program, say, without any extra-logical constructs). The content of the database is then $\mathrm{Cn}(D)$ where Cn represents what can be computed by the Prolog inference procedure from D. (How to characterise Prolog's inferences as a consequence relation Cn is not obvious and will be the subject of subsequent lecture.)

A query is $\quad \underline{x} : \alpha(\underline{x})?$
$\alpha(\underline{x})$ is any formula of $\mathcal{L}$ whose free variables are $\underline{x}$.
A query containing no free variables is a 'closed query'.

Answers to query $\underline{x} : \alpha(\underline{x})?$ on database D are those 'vectors' of constants $\underline{c}$ (or more generally ground terms if the language $\mathcal{L}$ contains function symbols) such that

$$\alpha(\underline{c}) \in \mathrm{Cn}(D) \qquad \text{i.e. } D \vdash \alpha(\underline{c})$$

There are *three* possible answers to a closed query α:

Yes	if $\alpha \in \mathrm{Cn}(D)$
No	if $\neg\alpha \in \mathrm{Cn}(D)$
Don't know	if $\alpha \notin \mathrm{Cn}(D)$ and $\neg\alpha \notin \mathrm{Cn}(D)$

In the terminology of belief sets, the three cases are the 'doxastic attitudes'. ('doxastic' means 'pertaining to belief'; 'epistemic' means 'pertaining to knowledge').

So there are three *basic kinds of belief change* for any given α:

- 'Don't know' to 'Yes' ('expansion')
- 'Yes' to 'Don't know' ('contraction')
- 'No' to 'Yes' ('revision')

Three basic kinds not six because of symmetry, e.g. 'Don't know' to 'No' for α is 'Don't know' to 'Yes' for $\neg\alpha$.

So how can we characterise and investigate

- $D^+\alpha$ — the expansion of D by α
- $D^-\alpha$ — the contraction of D by α
- $D^*\alpha$ — the revision of D by α

in terms of Cn and its properties?
This is the starting point for the AGM Theory of Belief Revision.
(No time for details, sorry.)

A theory or belief set $\mathrm{Cn}(D)$ is 'complete' iff for every formula α

- $\alpha \in \mathrm{Cn}(D)$ or $\neg\alpha \in \mathrm{Cn}(D)$.

Note that $\mathrm{Cn}(\emptyset)$ could also be a database. $\mathrm{Cn}(\emptyset) \neq \emptyset$, in general.
(For example, $p \vee \neg p \in \mathrm{Th}(\emptyset)$.)

Example

Consider the following (propositional) database D_{banker}:

$$b \to gm \vee am$$
$$b$$
$$p \to gm$$
$$\neg p$$

Take the content of the database to be $\mathrm{Th}(D_{\text{banker}})$.
Query: b ? Answer 'Yes', because $b \in \mathrm{Th}(D_{\text{banker}})$.
Query: p ? Answer 'No', because $\neg p \in \mathrm{Th}(D_{\text{banker}})$.
Query: gm ? Answer 'Don't know', because $gm \notin \mathrm{Th}(D_{\text{banker}})$ and $\neg gm \notin \mathrm{Th}(D_{\text{banker}})$.

Now suppose we take the content of the database to be $cwa_e(D_{\text{banker}})$ where

$$cwa_e(D) \quad \stackrel{\text{def}}{=} \quad \mathrm{Th}(D \cup \{\neg\alpha \mid \alpha \in atoms(\mathcal{L}) \text{ and } \alpha \notin D\})$$

(What kind of consequence relation is this cwa_e? We will examine its properties later in the course, but it's obvious it is supraclassical $\mathrm{Th}(D) \subseteq cwa_e(D)$ and also 'closed under truth-functional consequence' $\mathrm{Th}(cwa_e(D)) \subseteq cwa_e(D)$.)

Query: b ? Answer 'Yes', because $b \in cwa_e(D_{\text{banker}})$.
Query: p ? Answer 'No', because $\neg p \in cwa_e(D_{\text{banker}})$.
Query: gm ? Answer 'No', because $\neg gm \in cwa_e(D_{\text{banker}})$.

But notice that in this database we have:

$$\{gm \vee am, \neg gm, \neg am\} \subseteq cwa_e(D_{\text{banker}})$$

Inconsistent!! So in fact 'Yes' and 'No' are *both* correct answers to all (closed) queries.
(We might say that $cwa_e(D_{\text{banker}})$ is not a very good database.)

Consistency and integrity

A database $\mathrm{Cn}(D)$ is inconsistent iff there is some formula α such that both

- $\alpha \in \mathrm{Cn}(D)$ and $\neg\alpha \in \mathrm{Cn}(D)$.

So a formula α is consistent with a database $\mathrm{Cn}(D)$ when

- $\neg\alpha \notin \mathrm{Cn}(D)$

(Actually, this definition of inconsistency is *not as self-evident* as it might appear. For non-monotonic consequence relations, this notion of consistency does not work well and has to be adjusted. I will defer further discussion until we look at properties of non-monotonic consequence relations later in the course. Even then I might not cover it.)

Integrity constraints

There is also the (different) notion of database *integrity*: certain databases, even if consistent, are not wanted. We might say, for example, that a database is not wanted if it gives a person's address but without the postcode.

One common approach (in databases) is to represent integrity constraints as formulas of the language $\mathcal{L}$. And then one possible definition (the 'entailment' or 'theoremhood' definition of integrity constraint satisfaction) is to say that database D *satisfies* set of integrity constraints IC iff

- for every $\alpha \in IC$, $\alpha \in \mathrm{Cn}(D)$, i.e., $IC \subseteq \mathrm{Cn}(D)$

Another possibility, and another common definition, is to require only that integrity constraints IC are consistent with D: according to the 'consistency' definition, database D satisfies integrity constraints IC when

- $D \cup IC$ is consistent, i.e., $D \nvdash \neg \bigwedge(IC)$

The two definitions of integrity constraint satisfaction are different, because in general $\alpha \notin \mathrm{Cn}(D)$ is not equivalent to $\neg\alpha \in \mathrm{Cn}(D)$.
These treatments of integrity constraint only make sense, however, for particular special kinds of databases/knowledge bases: those that are 'complete', or nearly 'complete' (in some appropriate sense of 'nearly', which we put to one side for now).

The **more general** form of integrity constraint has the form:

- if $\alpha \in \mathrm{Cn}(D)$ then $\beta \in \mathrm{Cn}(D)$

This more general form is **not expressed** in the database language $\mathcal{L}$: it is a *metalevel* statement **about** the database. It subsumes the other kinds ('theoremhood/entitlement' and 'consistency') as special cases. There are also some *modal* languages for expressing such constraints.

Note that (in general)

- if $\alpha \in \mathrm{Cn}(D)$ then $\beta \in \mathrm{Cn}(D)$

is **not** equivalent to

- $(\alpha \to \beta) \in \mathrm{Cn}(D)$

(Simple example: consider an empty database. It won't be exactly empty, of course, because usually $\mathrm{Cn}(\emptyset) \neq \emptyset$. Suppose $p \notin \mathrm{Cn}(\emptyset)$. The integrity constraint $p \to q$ is not satisfied if $(p \to q) \notin \mathrm{Cn}(\emptyset)$. But even in that case, the metalevel integrity constraint 'if $p \in \mathrm{Cn}(D)$ then $q \in \mathrm{Cn}(D)$' *is* satisfied.)

Why is it then that some authors take the strongest ('theoremhood') definition of integrity constraint satisfaction while others take the weakest ('consistency') definition, without apparently noticing that, in general, neither does what they want? Because they are (usually) dealing with a *special case*, often without stating this explicitly.

We can summarise like this. Assume that Cn has the property that $D \subseteq \mathrm{Cn}(D)$ and also that it is 'closed under truth-functional consequence': $(\mathrm{Th}(\mathrm{Cn}(D)) \subseteq \mathrm{Cn}(D))$. Equivalently: if $D \vdash X$ and $X \vdash_{PL} Y$ then $D \vdash Y$. Both are very reasonable properties. Then:

$$\underset{\textit{weakest}}{\neg(\alpha \to \beta) \notin \mathrm{Cn}(D)} \xleftarrow{D \text{ consistent}} \text{if } \alpha \in \mathrm{Cn}(D) \text{ then } \beta \in \mathrm{Cn}(D) \longleftarrow \underset{\textit{strongest}}{(\alpha \to \beta) \in \mathrm{Cn}(D)}$$

If $\mathrm{Cn}(D)$ is *complete* (and consistent), then the three definitions of integrity constraint satisfaction coincide. Because then:

$$\underset{\textit{weakest}}{\neg(\alpha \to \beta) \notin \mathrm{Cn}(D)} \xrightarrow{D \text{ complete}} \text{if } \alpha \in \mathrm{Cn}(D) \text{ then } \beta \in \mathrm{Cn}(D) \xrightarrow{D \text{ complete}} \underset{\textit{strongest}}{(\alpha \to \beta) \in \mathrm{Cn}(D)}$$

(An exercise to check this is on the tutorial sheet.)

Example

Consider the following database DB consisting of a set of ground unit clauses:

```
has-office-in(IBM, Winchester)   city(Winchester)
has-office-in(IBM, London)       city(London)
has-office-in(IBM, Paris)        city(Paris)
has-office-in(MBI, London)       city(NewYork)

capital-city(London)             company(IBM)
capital-city(Paris)              company(MBI)
```

For this example we take the content of the database to be Th(DB).

Q1. x : `has-office-in(IBM,`x`)`?
has answers
$x =$ `Winchester`
$x =$ `London`
$x =$ `Paris`
Why? Because `has-office-in(IBM,`x`)` $\in$ Th(DB) for
$x =$ `Winchester`
$x =$ `London`
$x =$ `Paris`.

Q2. $\exists x$ `has-office-in(IBM,`x`)` ?
gets answer 'Yes' because $\exists x$ `has-office-in(IBM,`x`)` $\in$ Th(DB).

Notice that Q2 is a closed ('yes/no') query. Although it contains a variable x, this variable is not 'free': it is existentially quantified.

Q3. $\exists x$ (`has-office-in(IBM,`x`)` $\wedge$ `capital-city(`x`)`) ?
gets answer 'Yes' because $\exists x$ (`has-office-in(IBM,`x`)` $\wedge$ `capital-city(`x`)`) $\in$ Th(DB).

Q4. x : `has-office-in(`x`,NewYork)` ?
gets answer 'No solution' because there is no value of x such that `has-office-in(`x`,NewYork)` $\in$ Th(DB).

Q5. $\exists x$ `has-office-in(`x`,NewYork)` ?
gets answer 'Don't know'. Why? Because $\exists x$ `has-office-in(`x`,NewYork)` $\notin$ Th(DB) and $\neg\exists x$ `has-office-in(`x`,NewYork)` $\notin$ Th(DB).

Q6. `has-office-in(IBM,NewYork)` ?
gets answer 'Don't know'. Why? Similar reason as in Q5.

Relational database: model or theory?

As the previous example illustrates, a relational database can be viewed either

- as a set of relations $\mathcal{M}_{DB}$ (a *model*)
- as a set of (atomic) formulas DB, together with some notion of consequence Cn (a 'theory').

But: this notion of consequence Cn cannot be just truth-functional consequence Th: $\mathcal{M}_{DB}$ contains some implicit *negative* information that is not in Th(DB).

Question: can we make this negative information explicit? In other words, can we find an extra set of additional assumptions (formulas) X such that, for all α:

$$\mathcal{M}_{DB} \models \alpha \text{ iff } \alpha \in \text{Th}(DB \cup X)$$

X will have to add enough constraints to ensure that $DB \cup X$ has exactly one model (up to isomorphism), which is $\mathcal{M}_{DB}$.

It is possible to do that but I will leave the details for now. They will come out as a special case of some more general properties of logic programs and related formalisms.

The point is: the two-valued model $\mathcal{M}_{DB}$ has only true/false — no 'don't know'. As a representation, it embodies a form of so-called 'closed world assumption'.

Closed World Assumptions

In databases, and many forms of knowledge representation, it is common to make some kind of (often implicit) 'Closed World Assumption': what is not explicitly in the database is assumed to be false.

But beware: this statement of 'closed world assumptions' is **very imprecise**. There are **many different** ways of formalising Closed World Assumptions, with quite **different** properties. Here is a very simple example.

Consider again the earlier database DB, but instead of $\mathrm{Th}(DB)$ let's take the content of the database to be $cwa_e(DB)$, where (as before)

$$cwa_e(D) \quad \stackrel{\text{def}}{=} \quad \mathrm{Th}(D \cup \{\neg\alpha \mid \alpha \in atoms(\mathcal{L}) \text{ and } \alpha \notin D\})$$

(What kind of consequence relation is this cwa_e? We will examine its properties later in the course, but it's obvious it is supraclassical $\mathrm{Th}(D) \subseteq cwa_e(D)$ and also 'closed under truth-functional consequence $\mathrm{Th}(cwa_e(D)) \subseteq cwa_e(D)$.)

Now we get, for example:
Q6. `has-office-in(IBM,NewYork)` ?
gets answer 'No'. Why? Because $\neg$`has-office-in(IBM,NewYork)` $\in cwa_e(DB)$.

But
Q5. $\exists x$ `has-office-in(`x`,NewYork)` ?
still gets answer 'Don't know'. Why? Because $\exists x$ `has-office-in(`x`,NewYork)` $\notin cwa_e(DB)$ and $\neg\exists x$ `has-office-in(`x`,NewYork)` $\notin cwa_e(DB)$.

Notice that this database (or rather the consequence relation cwa_e) is **non-monotonic**:

- $\neg$`has-office-in(IBM,NewYork)` $\in cwa_e(DB)$

Add the extra fact `has-office-in(IBM,NewYork)` to DB:

- $\neg$`has-office-in(IBM,NewYork)` $\notin cwa_e(DB \cup \{$`has-office-in(IBM,NewYork)`$\})$

Notice that

(1) We haven't yet constructed the additional axioms X alluded to on the previous page that ensure $DB \cup X$ has a unique model. For example, we can't derive certain *quantified formulas* that are true in the model.

(2) This construction (sometimes called the 'Generalised Closed World Assumption' in the literature) only works for certain *very restricted* forms of database. See e.g. the D_{banker} example given earlier ($b \rightarrow gm \vee am$, etc) where $cwa_e(D_{\text{banker}})$ comes out as inconsistent.

More Closed World Assumptions: Examples

'Closed World Assumption' is a very loose term. There are many different versions. Here are some, just for the sake of some examples.

First, the one we considered earlier

$$cwa_e(D) \quad \stackrel{\text{def}}{=} \quad \mathrm{Th}(D \cup \{\neg\alpha \mid \alpha \in atoms(\mathcal{L}) \text{ and } \alpha \notin D\})$$

Suppose we try the following new consequence relation:

$$cwa(D) \quad \stackrel{\text{def}}{=} \quad \mathrm{Th}(D \cup \{\neg\alpha \mid \alpha \in atoms(\mathcal{L}) \text{ and } \alpha \notin \mathrm{Th}(D)\})$$

Or what about this alternative version?

$$cwa^*(D) \quad \stackrel{\text{def}}{=} \quad \mathrm{Th}(D \cup \{\neg\alpha \mid \alpha \in atoms(\mathcal{L}) \text{ and } \alpha \notin cwa^*(D)\})$$

Both of these last two resemble constructions found in non-monotonic logics.
We will examine properties of consequence relations such as these later in the course.

Notice that in these examples we have *non-monotonic* consequence (cwa, cwa^*) defined in terms of classical *monotonic* consequence. This is also a common feature of many non-monotonic reasoning systems.