Propositional Logic
Programs / Databases

Recall our example *conjunctive normal form* (CNF) formula:

$$\begin{array}{ll} \neg a \lor b & \neg b \lor \neg f \lor h \\ \neg a \lor c & \neg c \lor \neg d \lor h \\ \neg b \lor d \lor e & \neg e \lor \neg g \lor h \\ \neg c \lor f \lor g & a \end{array}$$

This is written in the common shorthand for CNF: There are implicit $\land$'s between the clauses.

A *clause* is of the form $l_1 \lor \ldots \lor l_k$, in which each l_i is a positive occurrence of a proposition (e.g., a) or a negative occurrence of a proposition (e.g., $\neg a$).

Oddly, let us call such a CNF formula as above a *program*, or even a *database*. It will become clearer as we progress why these names really are not so odd. Denote the program above as $\mathcal{P}$.

Propositional Logic
Queries

Prove h from $\mathcal{P}$.

In other words, we are asking the *query*, is h *true*, given that $\mathcal{P}$ is *true*?

What does this query mean? It means for us to show that h logically follows from $\mathcal{P}$.

There are several approaches to do this.

- **Model Theory Approach:** Show that in any situation in which $\mathcal{P}$ is *true*, h is also *true*.
- **Proof Theory Approach:** Show that there is a sequence of *inference* steps that lead from the premise $\mathcal{P}$ to the conclusion h.

The Example Simplified

Let us simplify our example $\mathcal{P}$ some so we do not have to work as hard.

$$\begin{array}{ll} \neg a \lor b & \neg b \lor \neg f \lor h \\ \neg a \lor c & \neg c \lor \neg d \lor h \\ \neg b \lor d \lor e & \neg e \lor \neg g \lor h \\ \neg c \lor f \lor g & a \end{array}$$

Does a have to be *true* for $\mathcal{P}$ to be *true*? Clearly yes. If a were *false*, $\mathcal{P}$ is necessarily *false*. So a is *true* in all situations (in which $\mathcal{P}$ is *true*). We also easily see then that b is *true* and that c is *true*.

So for any clause that contains a positive occurrence of a (or b or c), we can drop that clause from $\mathcal{P}$ since we now know that clause is *true*.
For any clause that contains $\neg a$ (or $\neg b$ or $\neg c$), we can drop the $\neg a$ (or $\neg b$ or $\neg c$) from it since we know $\neg a$ (and $\neg b$ and $\neg c$) is *false*. (We have to keep the rest of the clause though, because it still could be either *true* or *false*.

Thus our modified program, $\mathcal{P}'$, is

$$\begin{array}{ll} d \lor e & \neg d \lor h \\ f \lor g & \neg e \lor \neg g \lor h \\ \neg f \lor h & \end{array}$$

$\mathcal{P}$ and $\mathcal{P}'$—with a, b, and c as *facts* in $\mathcal{P}'$—are logically equivalent.

Truth Tables (Models)

We can guess for each proposition whether it is *true* or it is *false*, and see whether that makes $\mathcal{P}$ overall *true* or *false*, with respect to these guesses.

Each possible truth assignment—having assigned each proposition (e.g., $a, \ldots, h$) to *true* or to *false*—is called an *interpretation*.

Any interpretation that renders $\mathcal{P}$ *true* is called a *model* of $\mathcal{P}$.

This is really (so far) just the same as *truth tables*, which show up many places in C.S., E.E., and math.

- h *logically follows* from $\mathcal{P}$ iff h is *true* in every model of $\mathcal{P}$.
- Likewise, $\neg h$ *logically follows* from $\mathcal{P}$ iff h is *false* in every model of $\mathcal{P}$.

$\mathcal{P} \models h$ is the fancy way to write that h logically follows from $\mathcal{P}$.

In our example $\mathcal{P}$, there are eight propositions: $a, \ldots, h$. Therefore, there are 2^8, or 256, interpretations for $\mathcal{P}$.

In our $\mathcal{P}'$, there are just five propositions, $d, \ldots, h$, to interpret, so just 2^5, or 32, interpretations.

Truth Table for $\mathcal{P}'$

#	d	e	f	g	h	$\mathcal{P}'$
1	T	T	T	T	T	T
2	T	T	T	T	F	F
3	T	T	T	F	T	T
4	T	T	T	F	F	F
5	T	T	F	T	T	T
6	T	T	F	T	F	F
7	T	T	F	F	T	F
8	T	T	F	F	F	F
9	T	F	T	T	T	T
10	T	F	T	T	F	F
11	T	F	T	F	T	T
12	T	F	T	F	F	F
13	T	F	F	T	T	T
14	T	F	F	T	F	F
15	T	F	F	F	T	F
16	T	F	F	F	F	F

#	d	e	f	g	h	$\mathcal{P}'$
17	F	T	T	T	T	T
18	F	T	T	T	F	F
19	F	T	T	F	T	T
20	F	T	T	F	F	F
21	F	T	F	T	T	T
22	F	T	F	T	F	F
23	F	T	F	F	T	F
24	F	T	F	F	F	F
25	F	F	T	T	T	F
26	F	F	T	T	F	F
27	F	F	T	F	T	F
28	F	F	T	F	F	F
29	F	F	F	T	T	F
30	F	F	F	T	F	F
31	F	F	F	F	T	F
32	F	F	F	F	F	F

So nine interpretations—1, 3, 5, 9, 11, 13, 17, 19, and 21—render $\mathcal{P}'$ as *true*, and thus are models of $\mathcal{P}'$.

Note that h is *true* in all nine of the models. Therefore, h logically follows from $\mathcal{P}'$.

Since $\mathcal{P}$ and $\mathcal{P}'$ are logically equivalent, h logically follows from $\mathcal{P}$.

Models as Sets

Traditionally, an *interpretation* is a *subset* of the propositions to be considered *true*, and a *model* is an interpretation that renders $\mathcal{P}$ as *true*.

So the models in our example $\mathcal{P}'$ are

1. $\{d, e, f, g, h\}$
2. $\{d, e, f, h\}$
3. $\{d, e, g, h\}$
4. $\{d, f, g, h\}$
5. $\{d, f, h\}$
6. $\{d, g, h\}$
7. $\{e, f, g, h\}$
8. $\{e, f, h\}$
9. $\{e, g, h\}$

Any proposition that does not appear in a model then is considered to be *false with respect to* that model. For instance, d is *false* with respect to model #9.

Rewording then what it means to *logically follow*:

- h logically follows from $\mathcal{P}$ iff h is in *every* model of $\mathcal{P}$;
- likewise, $\neg h$ logically follows from $\mathcal{P}$ iff h is not in *any* model of $\mathcal{P}$.

Thus to find the set of all the propositions that logically follow from $\mathcal{P}$, find the interection of all $\mathcal{P}$'s models.

For $\mathcal{P}'$, that is $\{h\}$.

For our example $\mathcal{P}$, that would be $\{a, b, c, h\}$.

Consistency

Is it ever possible that $\mathcal{P} \models a$ *and* that $\mathcal{P} \models \neg a$?

This seems bizarre, but it is possible!

It works by our definitions if $\mathcal{P}$ has no models.

Consider our example $\mathcal{P}$, but with the clause $\neg h$ added. This new $\mathcal{P}$ has no models.

A program $\mathcal{P}$ with no models is called *inconsistent*. $\mathcal{P}$ is called *consistent* otherwise.

Ideally, we would like to consider only consistent $\mathcal{P}$'s.

Unknown

Is it ever possible that $\mathcal{P} \not\models a$ and that $\mathcal{P} \not\models \neg a$?

This says that a does not logically follow from $\mathcal{P}$ *and* that $\neg a$ does not logically follow from $\mathcal{P}$.

Certainly this is possible.

This might seem bizarre initially, but really it is not so odd. It is just that a given $\mathcal{P}$ may not provide enough information to determine whether a is *true* or it is *false*.

In this case we would say that a is *unknown* with respect to $\mathcal{P}$.

For example, in our example $\mathcal{P}'$, d is *unknown*.

Refutation
by Truth Tables

The notion of inconsistency gives us another method to show that something logically follows.

Add the negation of what we are trying to prove to $\mathcal{P}$, and show that the resulting $\mathcal{P}$ is inconsistent (that is, has no models).

For example, if we can show that there are no *models* of $\mathcal{P}' \cup \{\neg h\}$, then $\mathcal{P}' \models$ h.

#	d	e	f	g	$\mathcal{P}' \cup \{\neg h\}$
1	T	T	T	T	F
2	T	T	T	T	F
3	T	T	T	F	F
4	T	T	T	F	F
5	T	T	F	T	F
6	T	T	F	T	F
7	T	T	F	F	F
8	T	T	F	F	F
9	T	F	T	T	F
10	T	F	T	T	F
11	T	F	T	F	F
12	T	F	T	F	F
13	T	F	F	T	F
14	T	F	F	T	F
15	T	F	F	F	F
16	T	F	F	F	F

(You should check this truth table as an exercise.)

Refutation (p.2)
by Truth Tables

So once again we have proven that h logically follows from $\mathcal{P}'$.

In our example for $\mathcal{P}' \models h$?, by refutation we only needed to look at 16 interpretations. We had to look at all 32 interpretations for $\mathcal{P}'$ before.

We shall extend this idea of refutation into a proof system.

Proof-by-refutation is a type of proof by contradiction.

Note that we did not learn whether $\mathcal{P}$ is consistent or not this way.

We happened to know already that our example $\mathcal{P}'$ is consistent, because we know from before that it has a model (actually, nine models).

What about the case when we do not know whether $\mathcal{P}$ is consistent?

Proof Theory

Another approach to find whether a logically follows from $\mathcal{P}$ is to apply a sequence of *logically sound* inference steps starting from $\mathcal{P}$ and ending with a.

The sequence of steps from $\mathcal{P}$ to a is called a *proof*, and serves to prove that a can be logically derived from $\mathcal{P}$.

$\mathcal{P} \vdash a$ is the fancy way to write this. This states that there exists a proof of a from $\mathcal{P}$.

There are a number of questions to address regarding our proof system.

First, what are the types of inference steps that are permitted? Once we have established that, we must address *soundness* and *completeness*.

- **Soundness**: For any $\mathcal{P}$ and a, if $\mathcal{P} \vdash a$, then $\mathcal{P} \models a$.
- **Completeness**: For any $\mathcal{P}$ and a, if $\mathcal{P} \models a$, then $\mathcal{P} \vdash a$.

$\vdash \equiv \models$
An Aside

First-order propositional logic is sound and complete. That is, anything provable in it is in fact *true*, and anything *true* with respect to it is provable.

It is one of the greatest mathematical results of the 20th century that first-order logic (predicate calculus) without arithmetic is sound and complete. (**Gödel's Completeness Theorem**)

It is arguably the greatest mathematical result of the 20th century that first-order logic (predicate calculus) with full arithmetic and second-order logic are necessarily incomplete. (**Gödel's Incompleteness Theorem**)

Refutation Proofs
with Resolution

For our CNF programs, remarkably there is *one* inference rule that will suffice: *resolution*.

(This is not exactly true. We would need more to be *complete* for CNF programs. However, it is all we will need for Datalog to come.)

The resolution step is as follows.

$$\frac{(a \vee l_1 \vee \ldots \vee l_k) \wedge (\neg a \vee l_{k+1} \vee \ldots \vee l_n)}{l_1 \vee \ldots \vee l_n}$$

in which each l_i is a positive or a negative occurrence of a proposition.

Refutation proof by resolution:

- Add the negation of what you are trying to prove. E.g., $\mathcal{P}' \cup \{\neg h\}$.
- Apply resolution steps until you reach the *empty clause*.

The empty clause (an *or*-clause with nothing in it) is equivalent to *false*. (Think about it.)

Refutation Proof by Resolution

of $\mathcal{P}' \cup \{\neg h\}$

Let us show that h logically follows from $\mathcal{P}'$ once again, this time with a refutation proof by resolution.

$\neg h$	$\neg d \vee h$
	$\neg d$
$\neg d$	$d \vee e$
	e
$\neg h$	$\neg f \vee h$
	$\neg f$
$\neg f$	$f \vee g$
	g
e	$\neg e \vee \neg g \vee h$
	$\neg g \vee h$
g	$\neg g \vee h$
	h
$\neg h$	h
	$\Box$

'$\Box$' is a fancy symbol used to denote the empty clause.

Thus, we have arrived at '$\Box$', or *false*. This is a contradiction. Therefore, h cannot be *false*, (equivalently, $\neg h$ cannot be *true*), so h must be *true* (with respect to $\mathcal{P}$).

Logic as a Database? Problems?

Horn Programs

What are the problems with using CNF propositional logic for our knowledge-bases / databases / programs?

1. **Looks really unnatural.**
2. **Allows for *meaningless* databases / programs (that is, that have *no* models).**
3. **Where's the data?!**

To address both points 1 and 2, we shall restrict the types of clauses permitted.

Horn clause: At most one positive proposition appears.

Horn clauses are more natural, and have a correspondence to database concepts.

Rule / View:	$a \leftarrow b, c.$	$(a \vee \neg b \vee \neg c)$
Fact:	$b.$	(b)
Query:	$\leftarrow a.$	$(\neg a)$

- Easy to read when rewrittin as implications.
- Every database / program is meaningful; that is, consistent! (*Really?!* ...)

Logic as a Database!

Datalog

We call a program that consists of just *rules* and *facts*—Horn clauses each with exactly one positive proposition—a *Datalog* database.

We write queries as Horn clauses that contain no positive proposition.

A query can be *evaluated* against a Datalog database as a resolution refutation proof.

---

query evaluation $\equiv$ **proof**

So how do we know an answer to a query (with respect to the database) is *correct*?

Its evaluation is *equivalent* to a proof that it is correct (that it is a logical consequence)!

We still need to address point 3, "Where's the data?!"

We shall need to use (first-order) *predicate calculus* logic instead of just (first-order) propositional logic.

Datalog with Resolution

Example

$$\begin{array}{l} \leftarrow a,\ d. \\ a \leftarrow b,\ c. \\ \hline \leftarrow b,\ c,\ d. \end{array}$$

This is just resolution in disguise.

$$\begin{array}{l} \neg a \vee \neg d \\ a \vee \neg b \vee \neg c \\ \hline \neg b \vee \neg c \vee \neg d \end{array}$$

Datalog Models
Always Consistent!

A Datalog database is always consistent. That is, any datalog database is guaranteed to have at least one model.

How do we know?

Consider the interpretation in which we assign *true* to every proposition. Next, consider each clause: There is exactly one positive proposition per clause in a Datalog database, *by definition*. Thus every clause is *true* with respect to the all-true interpretation, and thus the all-true interpretation is a model.

Of course the all-true model is not so interesting. . .but it does guarantee that any datalog database is consistent.

The Minimum Model

Datalog

Our example $\mathcal{P}'$ is not a datalog database because it has non-Horn clauses.

$\mathcal{P}'$ also has nine models.

Which of the nine models captures the "meaning" of $\mathcal{P}'$? No one of them, per se, but rather all of them collectively…The fact there are nine of them is confusing and headache-causing.

It may make sense to consider only the *minimal models.* That is, throw away any model that is a super-set of another.

When we do that for $\mathcal{P}'$, we are left with just four minimal models.

1. $\{d, f, h\}$
2. $\{d, g, h\}$
3. $\{e, f, h\}$
4. $\{e, g, h\}$

Better, but we still have multiple models…

Any Datalog database has exactly *one* minimal (hence, *minimum*) model.

That minimum model is equivalent to exactly the set of propositions that logically follow.

We consider this minimum model to be the *meaning* of the Datalog database.

The Minimum Model

Example

$$a \leftarrow b,\ c.$$

$$b.$$

This Datalog database has eight interpretations.

1. $\{\}$
2. $\{a\}$
3. $\boxed{\{b\}}$
4. $\{c\}$
5. $\underline{\{a, b\}}$
6. $\{a, c\}$
7. $\{b, c\}$
8. $\underline{\{a, b, c\}}$

The underlined interpretations are models.

The boxed interpretation is the minimum model.

Reasoning about Queries & Databases

Datalog and its logical foundations—model and proof theories—provide us with tools to address other general questions about queries and databases.

- Given two Datalog queries, are they equivalent with respect to the database?

  That is, must they evaluate to the same answers?
- Does there *exist* a query of a given question we have in mind?

  That is, is the question even *askable* in Datalog? With this database?
- Given to Datalog databases, do they represent the same data?

  Is any question possible to state for one of them also possible to state for the other one?

Query equivalence—and more generally, *query containment*—is important many places.

For instance, the rewrite query optimizer must guarantee that the rewritten query is equivalent to the original query.

Containment
Example

$\mathcal{P}$:

$$a \leftarrow b,\ c. \qquad e \leftarrow b,\ c, f.$$
$$a \leftarrow d.$$

Does $\mathcal{P} \models e \rightarrow a$? Why or why not?

If not, how should we restrict the semantics for $\mathcal{P}$ so that we could infer $e \rightarrow a$?

The Move from Propositional Logic to Predicate Calculus

- Add arguments to propositions.

  Now call them *predicates*.
- Add logical variables.

  (For use in rules and in queries.)
- Add quantifiers for the variables: $\forall$ and $\exists$.

E.g.,

$$grandmother\ (GM,\ X) \leftarrow mother\ (GM,\ P),\ parent\ (P,\ X).$$

By convention, we shall write *variables* beginning with a capital letter, and *constants*—that is, data values—beginning with a lower-case letter or in single quotes.

Predicate Calculus
Horn Clauses

$$grandmother\,(GM,\ X) \leftarrow mother\,(GM,\ P),$$
$$parent\,(P,\ X).$$

is just a clause, as before.

Each variable is understood to be within the scope of a $forall$-quantifier.

So the clause above is shorthand for

$$\forall GM, X, P\,(grandmother\,(GM,\ X) \lor \neg mother\,(GM,\ P)$$
$$\lor \neg parent\,(P,\ X))$$

which is equivalent to

$$\forall GM, X\,(grandmother\,(GM,\ X)\ \leftarrow$$
$$\exists P\,(mother\,(GM,\ P) \land parent\,(P,\ X)))$$

Datalog permits only universal-quantified clauses. Thus no explicit existential-quantification is allowed.

Datalog Database versus "Prolog" Program

We generally call a Horn-clause predicate calculus "theory" that we have written down a *logic program*.

The Prolog programming language's syntax looks just like this.

So when do we call it a *Datalog database* instead?

If it uses *logical function symbols*, it is considered a *program*. If it does not, it is considered a *database*.

This is the logical distinction between them.

Logical function symbols??

This is essentially a data-structure, such as a list or record, that we could use as an argument to a predicate instead of just a simple value.

- $grandmothers\ (\boxed{[lallage,\ ruby,\ sally]},\ parke)$
- $product\ (\#13,\ \boxed{widget\ (a,\ b)},\ \$23.50)$

Function symbols are needed for arithmetic. (We usually add a limited form of arithmetic to a fuller Datalog.)

Safeness

We only permit *safe* clauses in Datalog.

A clause is *safe* iff every variable that appears in the positive atom (that is, on the left-hand side of the '$\leftarrow$') also appears in a negative atom (that is, on the right-hand side of the '$\leftarrow$'). Thus,

$$h\,(X_1,\ \ldots,\ X_k)\ \leftarrow\ b_1\,(Y_1,\ \ldots,\ Y_{j_1}), \ldots, b_n\,(Y_{j_{n-1}+1},\ \ldots,\ Y_{j_n}).$$

is safe if

$\{X_1, \ldots, X_k\} \subseteq \{Y_1, \ldots, Y_{j_n}\}$

E.g.,

$$h\,(X,\ Y) \leftarrow b\,(X).$$

is not safe.

Note that *facts* in Datalog cannot have variables. A fact with variables is not safe, by definition.

Predicate Calculus
Models

The model semantics remains essentially the same as for the propositional case, but now is much more complex to think about.

In particular, now models can be *infinite*!

For Datalog databases (DDBs), there are ways to limit our focus to a finite set of interpretations / models (e.g., the Herbrand interpretations / models). However, there can be *many* of them.

Predicate Calculus
Proof Theory

The proof theory remains essentially the same as for the propositional case, but now is much more complex to think about.

Resolution remains a sound and complete inference rule.

We have to add *unification*: a variable can become bound to a constant (a value).

Grandparent Database
Datalog

A simple Datalog database:

$$grandmother\,(GM,\ X) \leftarrow mother\,(GM,\ P),\ parent\,(P,\ X).$$

$$grandfather\,(GF,\ X) \leftarrow father\,(GF,\ P),\ parent\,(P,\ X).$$

$$parent\,(M,\ X) \leftarrow mother\,(M,\ X).$$

$$parent\,(F,\ X) \leftarrow father\,(F,\ X).$$

$mother\,(judith,\ parke).$	$father\,(blan,\ parke).$
$mother\,(ruby,\ judith).$	$father\,(alvin,\ judith).$
$mother\,(lallage,\ blan).$	$father\,(albert,\ blan).$

Two queries for the database:

$\leftarrow grandmother\,(G,\ parke).$	$\leftarrow grandmother\,(lallage,\ X).$
G = ruby;	X = parke;
G = lallage;	no
no	

Siblings?

How would we write a rule for siblings?

$$\begin{aligned} sibling\,(X,\ Y) \leftarrow\ & parent\,(P,\ X), \\ & parent\,(P,\ Y), \\ & X \neq Y. \end{aligned}$$

Brother? I.e., B is the brother of X.

$$\begin{aligned} brother\,(B,\ X) \leftarrow\ & parent\,(P,\ B), \\ & parent\,(P,\ X), \\ & male\,(B), \\ & B \neq X. \end{aligned}$$

Sister? I.e., S is the sister of X.

$$\begin{aligned} sister\,(B,\ X) \leftarrow\ & parent\,(P,\ B), \\ & parent\,(P,\ X), \\ & female\,(B), \\ & B \neq X. \end{aligned}$$

Ancestor?
Recursion!

$$ancestor\,(A,\ X) \leftarrow parent\,(A,\ X).$$

$$\begin{aligned} ancestor\,(A,\ X) \leftarrow\ & parent\,(A,\ B), \\ & ancestor\,(B,\ X). \end{aligned}$$

We have *recursion* in Datalog? Of course.

Nothing in our definitions forbids it.

And recursion is a very useful tool in defining rules and queries.

Cousin?

Negation. . .

$$\begin{aligned} cousin\,(X,\ Y) \leftarrow\ & parent\,(A,\ X), \\ & parent\,(B,\ Y), \\ & sibling\,(A,\ B). \end{aligned}$$

Wait! Siblings are *not* cousins.

$$\begin{aligned} cousin\,(X,\ Y) \leftarrow\ & parent\,(A,\ X), \\ & parent\,(B,\ Y), \\ & sibling\,(A,\ B), \\ & \textsf{not}\ sibling\,(X,\ Y). \end{aligned}$$

However, this is not Datalog. What is that "not"?

Adding negation to Datalog is going to be a challenge. . .

Inferring the Negative
Datalog

Is there ever a Datalog program $\mathcal{P}$ and an atom a such that $\mathcal{P} \models \neg a$?

No! Recall the all-true interpretation is always a model of any Datalog database (DDB).

We pulled that trick so that *every* DDB is guaranteed to be consistent (that is, to have a model).

So how can we ask negative questions in Datalog?

E.g., Is parke not a student?

Can we?

The Closed World Assumption (CWA)
Datalog

Model-theoretic

Only accept the minimum model, $\mathcal{M}$. If $a \notin \mathcal{M}$, then say that $\neg a$ is *true* (or equivalently, that a is *false*).

Proof-theoretic

Negation-as-Finite-Failure (NAFF).

If $\mathcal{P} \not\vdash a$, then say that $\neg a$ is *true*.

NAFF is sound with respect to safe Prolog / Datalog, but it is not *complete*.

Full first-order predicate calculus is undecidable.

Okay. This allows negation in queries.

How about *within* programs themselves, like with *cousin*?

What does Datalog have that SQL doesn't?

- a clear semantics
- recursion (until recently!)
- is easier to write and think about (?)

What does SQL have that Datalog doesn't?

- aggregation
- negation! (except)
- NULLs

Also SQL is a real language and Datalog is a play language, so they are hard to compare in this sense.