Last class

Knowledge representation: encoding ontologies.

An ontology is a collection of *categories* and *relations*.

First unit: focusing on ontologies that are *sets*, especially *objects*.

Today: zooming out with propositional logic.

Propositions

The atomic (indivisible) unit is the *proposition*:

$$p \triangleq \text{“Paris is the capital of France.”}$$
$$q \triangleq \text{“Mice chase elephants.”}$$

$$\begin{aligned}
\texttt{brain}(p) = \texttt{brain}(\text{“Paris is the capital of France.”}) &= \textit{true} \\
\texttt{brain}(q) = \texttt{brain}(\text{“Mice chase elephants.”}) &= \textit{false}
\end{aligned}$$

Interpretation against a Knowledge Base

$$\texttt{brain}(p) = \texttt{brain}(\text{“Paris is the capital of France.”}) = \textit{true}$$

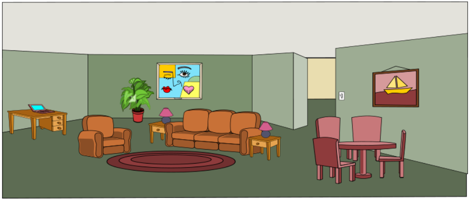
What's happening here:

1. Propositions get defined and stored somewhere.
2. Eventually we want to use or reason over propositions:
   2.1 Replace propositions with their definitions.
   2.2 Check definitions against what we know.

Replace with $\langle KB, \sigma, [\![\cdot]\!]\rangle$

- KB: Facts (a set of strings).
- σ: A set of propositions.
- $[\![\cdot]\!] \triangleq$ a function from propositions to $\{\textit{true}, \textit{false}\}$

Example



KB

The table has four chairs.

A photograph is on the wall.

The computer is on.

The computer battery is dead.

σ

$p \triangleq$ The table has four chairs.

$q \triangleq$ A painting is on the wall or a photograph is on the wall.

$r \triangleq$ The computer is not on.

$s \triangleq$ If the computer is on, then it is not dead.

$t \triangleq$ If the computer is on, then the battery is not dead.

$u \triangleq$ The plant is made of plastic.

Connectives

Propositions compose into *formulas* via *connectives*, which are *inductively* defined:

- *Atomic formulas*: $F \in \{p_1, p_2, \ldots\}$
- *Negation*: If F is a formula, then $\neg F$ is a formula.
- *Disjunction*: If F and G are formulas, then $F \vee G$ is a formula.

(These are the only *required* connectives, but for readability...)

- *Conjunction*: If F and G are formulas, then $F \wedge G$ is a formula.
- *Implication*: If F and G are formulas, then $F \rightarrow G$ is a formula.

You should be able to recognize syntactically valid formulas.

Clicker Question

Let the set of propositions be $\{p_i \mid i \in \mathbb{N}\}$ and a set of formulas be $\{F_i \mid i \in \mathbb{N}\}$.

Which of the following is *not* a formula?

A) $\neg((p_1 \to p_2) \to (\neg p_2 \to \neg p_1))$

B) $p_1, p_2 \to r$

C) $(p_1 \wedge p2) \to \neg(\neg p_1 \vee \neg p_2)$

D) $F_1 \wedge (p_1 \vee p_2)$

E) $F_1 \to F_2 \to F_3$

Equivalence?

$$F \wedge G \triangleq \neg(\neg F \vee \neg G)$$
$$F \rightarrow G \triangleq \neg F \vee G$$

- Right now, these are just *syntactic* rewrites:

$$\neg F \vee \neg(\neg G \vee \neg H)$$

- They also happen have an intuitive meaning.

Classically...

- All propositions have a *truth value* of 0 or 1.
- An assignment $\mathcal{A}$ is a mapping from propositions to truth values.
- We evaluate formulas down to truth values using assignment $\mathcal{A}$ (i.e., $[\![\cdot]\!]_{\mathcal{A}}$: formulas $\to \{0, 1\}$) and the rules:

F_1	F_2	$\neg F_1$	$F_1 \vee F_2$	$F_1 \wedge F_2$	$F_1 \to F_2$

Truth tables

Let $F = p_1 \vee (p_2 \wedge \neg p_2)$ and $\mathcal{A} = \{p_1 \mapsto 0; p_2 \mapsto 1\}$. Find $[\![F]\!]_{\mathcal{A}}$.

Inputs		Subexpressions		Output
p_1	p_2	$\neg p_2$	$p_2 \wedge \neg p_2$	$p_1 \vee (p_2 \wedge \neg p_2)$

You should know how to evaluate formulas using truth tables and identify inputs, subexpressions, and outputs.

Syntactic vs. Semantic true and false

Observe: $[\![p_2 \wedge \neg p_2]\!]_{\mathcal{A}} = 0$ for all possible $\mathcal{A}$.

It would be nice to replace $p_2 \wedge \neg p_2$ with 0 in $p_1 \vee (p_2 \wedge \neg p_2)$:

$$p_1 \vee 0$$

Problem: This is not valid formula syntax!
Solution: add two special values to our Atomic formulas:

$$F \in \{\top, \bot, p_1, p_2, \ldots\}$$

such that:

$$[\![\top]\!]_{\mathcal{A}} = 1 \quad [\![\bot]\!]_{\mathcal{A}} = 0$$

Equivalence

$$F \equiv G \triangleq [\![F]\!]_{\mathcal{A}_1} = [\![G]\!]_{\mathcal{A}_1} \land \cdots \land [\![F]\!]_{\mathcal{A}_n} = [\![G]\!]_{\mathcal{A}_n}$$

Two formulas F and G are equivalent if and only if they evaluate to the same truth value for all *suitable* assignments.

- A *suitable* assignment contains truth values for all of the propositions (atomic formulas) used in a formula.
- How many possible assignments?

You should know how to test whether two formulas are equivalent.

Equivalence example: disjunction and implication

Check syntactic rewrite against "intuitive" definitions:

Inputs		Output 1	Subexpressions		Output 2
p_1	p_2	$p_1 \wedge p_2$	$\neg p_1$	$\neg p_2$	$\neg(\neg p_1 \vee \neg p_2)$
p_1	p_2	$p_1 \rightarrow p_2$	$\neg p_1$		$\neg p_1 \vee p_2$

Scenarios important enough to have their own names

1. If $\mathcal{A}$ is suitable for F and $[\![F]\!]_{\mathcal{A}} = 1$, then we say $\mathcal{A}$ *models* F, written $\mathcal{A} \models F$.

2. If all possible $\mathcal{A} \models F$, then F is *valid* or a *tautology*.

3. If there are no suitable assignments $\mathcal{A}$ such that $\mathcal{A} \models F$, then F is *unsatisfiable* or a *contradiction*.

4. If there is *at least one* possible $\mathcal{A}$ such that $\mathcal{A} \models F$, then F is *satisfiable*.

You should know the definitions of valid/tautology, unsatisfiable/contradiction, and satisfiable.

Boolean satisfiability (SAT)

Problem: find an assignment $\mathcal{A}$ such that $[\![F]\!]_{\mathcal{A}} = 1$.
Naive solution:

1. Enumerate all possible assignments for F, $\mathbb{A} = \{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$.
2. For each $\mathcal{A}_i \in \mathbb{A}$, if $[\![F]\!]_{\mathcal{A}_i} = 1$, return $\mathcal{A}_i$.
3. If no such assignment exists, return UNSAT.

Let $F = p_1 \vee (p_2 \wedge \neg p_2)$. Find a satisfying assignment for $\mathcal{A}$:

p_1	p_2	$\neg p_2$	$p_2 \wedge \neg p_2$	$p_1 \vee (p_2 \wedge \neg p_2)$
0	1	0	0	0

(3-)SAT is in NP

Boolean satisfiability is the canonical NP-complete problem:

- NP-*Complete*: it requires as many resources as the most resource-hungry problems in NP, but no more.
- Solutions can be checked in polynomial time (function of inputs, i.e., propositions).
- Solutions may need up to exponential time to be found.

Conjunctive Normal Form (CNF)

We convert formulas to a normalized form to make things easier.

$$(p_1 \vee p_2 \vee p3) \wedge (p_2 \vee p_4 \vee \neg p_5) \wedge (\neg p_1 \vee p_6)$$

$$(\neg p_1 \vee \neg p_2) \wedge (\neg p_2 \vee \neg p_3 \vee p_4) \wedge p_5$$

1. Rewrite $F \to G$ as $\neg F \vee G$.
2. Push all negation to atomic formulas using rewrites (future inference rules, theorems, and axioms).
3. Apply distributivity laws.

You do not need to know the specifics, just that it is algorithmic.

Common rewrite rules

Conversion to CNF uses common rewrite rules:

- $\neg\neg F = F$ (double negation)
- $\neg(F \wedge G) = \neg F \vee \neg G$ (deMorgan's)
- $\neg(F \vee G) = \neg F \wedge \neg G$ (deMorgan's)
- $F \wedge (G \vee H) = (F \wedge G) \vee (F \wedge H)$ (distributive)
- $F \vee (G \wedge H) = (F \vee G) \wedge (F \vee H)$ (distributive)

You should know and recognize these transformations.

Horn Formulas

A formula is said to be a *Horn formula* iff it is in CNF and each *clause* has at most one positive atomic formula (proposition).

$$(p_1 \wedge p_2 \wedge p3) \vee (p_2 \wedge p_4 \wedge \neg p_5) \vee (\neg p_1 \wedge p_6)$$

$$(p_1 \vee p_2 \vee p3) \wedge (p_2 \vee p_4 \vee \neg p_5) \wedge (\neg p_1 \vee p_6)$$

$$(\neg p_1 \vee \neg p_2) \wedge (\neg p_2 \vee \neg p_3 \vee p_4) \wedge p_5$$

There exists an efficient algorithm for Horn-SAT.

You should be able to recognize Horn formulas.

Complexity vs. Algorithms vs. AI

How is this an AI problem?

- Complexity theory: categorize into *equivalence classes*.
  - *If we had an efficient solution for 3-SAT, what other problems could we use it to solve?*
  - Tries to do this for *arbitrary* syntactically correct formulas.
- Algorithms: find a solution to an arbitrary problem instance.
  - *For problems of a certain form, can we find an efficient algorithm that is correct?*
  - Tries to do this for *specific* subclasses of syntactically correct formulas.
- AI: find a solution that is correct/fast for most of the problems, most of the time.
  - *Can we identify* **common problem features** *to help us solve our problems?*
  - Okay to sometimes be slower and/or incorrect.

Using problem features

One step toward "looking at the data." Example in SAT:

- "Unit clauses."
- Ordering by clause size.
- *Literals* (an atomic formula or its negation) that are always "the same."
- Ordering by number of *positive literals* (an atomic formula).

Key difference from machine learning: "data" here are the **problem instances**, *not the inputs.*

Key algorithmic technique: early stopping.
You should be able to recognize the difference between features of the input data and features of a problem space.

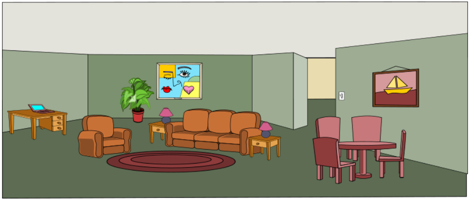
Semantics: giving meaning to syntax

Recall: $[\![\cdot]\!] \triangleq$ a function from propositions to $\{true, false\}$

Now: $[\![\cdot]\!]_{\langle KB,\sigma\rangle}$: formulas $\rightarrow \{1,0\}$

F	$[\![F]\!]$
p_i	if $\sigma(p_i) \in KB$ then 1 else 0
$\neg p_i$	if $\sigma(p_i) \notin KB$ then 1 else 0
$F \vee G$	if $[\![F]\!]$ then 1 else $[\![G]\!]$
$F \wedge G$	if $[\![F]\!]$ then $[\![G]\!]$ else 0
$F \rightarrow G$	if $[\![F]\!]$ then $[\![G]\!]$ else 1

Example



KB

The table has four chairs.

A photograph is on the wall.

The computer is on.

The computer battery is dead.

σ

$p \triangleq$ The table has four chairs.

$q \triangleq$ A painting is on the wall or a photograph is on the wall.

$r \triangleq$ The computer is not on.

$s \triangleq$ If the computer is on, then it is not dead.

$t \triangleq$ If the computer is on, then the battery is not dead.

$u \triangleq$ The plant is made of plastic.

Next class

Getting more expressive and using an ontology with predicate logic.