# Lecture 8

# Knowledge Representation and Logic

So far all the knowledge we know about the world is in the form of a state, which is more or less a black-box to our reasoning system.

We will learn a new way to represent the world, so that the reasoning system can do much more for us.

**Reference**:

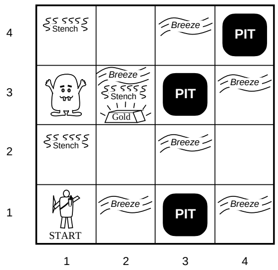
- Textbook Chapter 6

## Motivation

So far, to represent a state, we define a **new** structure.

- This works well when our problem is in a **very simple environment**.
- E.g., the whole state in the missionaries and cannibals problem can be represented just by 4 integers and a boolean.
- We will only query **very simple properties** of the state. E.g., what are the possible next state.
  For other problems, what is the value of the heuristic function or the evaluation function, is it the min's turn or the max's turn, etc.
- We can **easily encode** how to answer the queries as **functions**.
- Unluckily, the **real world is rarely this simple**.
  We study simple problems first so that we can focus on search algorithms. Now it is time to start worrying about data representation.

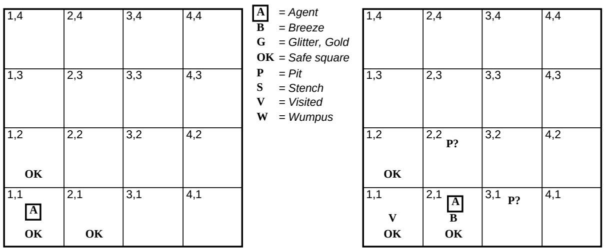
## Example problem: Wumpus World



- We are at the SW corner of a 4 × 4 "cave", like the one on the right.
- There is a nasty animal called the **wumpus** in one of the rooms, and will eat anyone entering the room.
  But you can shoot an arrow to kill it.
- Each room has a small probability to contain bottomless **pit**. Entering these room results in death.
- But the maze has **a piece of gold**, and we want to collect it.
- The world is less hazardous than it sounds: when you are real close to (1 room besides) a room with pit or the wumpus, you will sense it. You feel a breeze and a stench respectively.
- We have **no prior knowledge** to where are the pits, wumpus and gold.

## Solving the problem ourselves

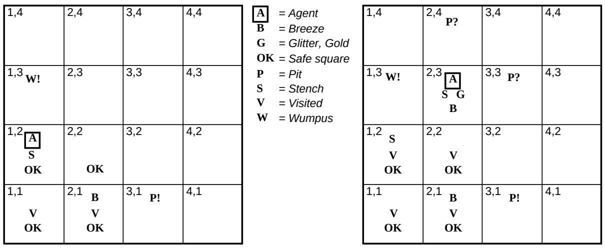
To see **what queries we would make** to our state, let's try solving the problem ourselves—using the cave we just see.



(a) (b)

- Initially, we don't sense anything, meaning 1,2 and 2,1 are safe.
- We enter 2,1, feeling a breeze, and conclude a pit is at either 2,2 or 3,1.

## Solving the problem ourselves (cont'd)



(a) (b)

- Then we try 1,2, feeling a stench, meaning the wumpus is at 1,3 or 2,2.
- Combining the two piece of knowledge, now 2,2 becomes safe. We enter that, and found that 2,3 and 3,2 are both safe.
- Eventually we go to 2,3 and found the gold.

## Automatic reasoning

- What are our **percepts**? We get percepts about whether we feel a stench, a breeze, or the gold on a new square we enter.
- What are our **queries**? We ask **whether a square is known to be safe.**
- What our state contains? It will contain things like "Room 1,2 contains no pit", "Either room 2,2 or room 1,3 contains a wumpus", etc.
- **Can we represent the state in a struct just as before**? Yes…
  We can have 4 matrices stating whether there is a stench, a breeze, known to contain no pit, and known to contain no wumpus.
- But it will require **a lot of programming efforts to reason** and update the state (e.g., to infer that there is a wumpus in a certain room, etc).
  Actually, the wumpus world is still "too easy" for coding all rules directly.
- Question: **Can we write a program which automate the reasoning?**

## Knowledge Representation

An automatic reasoning program must **“know” a lot of facts** before we can perform reasoning on what we know:

- What is perceived so far.

  Breeze at 1,2; Stench at 2,1, etc.
- The rule of the state.

  If breeze, then there is pit in at least one of the squares around.
  If there is no breeze, none of the squares around contains a pit.
  There is only 1 wumpus.
- Temporal information.

  Wumpus is found, but since the last action is to shoot, it is dead and the wumpus no longer pose problems.
- And many other background knowledge.

We will need to be able to **represent an open-ended variety of facts**, but in a way **easily accessible by the reasoning program**.

## Knowledge representation and knowledge base

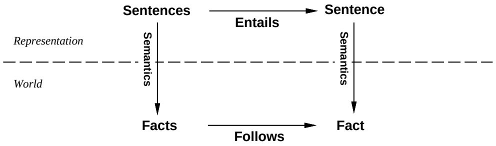
- Our knowledge will be represented as **sentences** in the computer. These sentences are in a special **knowledge representation language** (KRL). The sentences that the program knows form a **knowledge base** (KB).
- Just like any other language, the **syntax** of a KRL determines **what sentences look like** (i.e., what can be said to be sentences).
- We will use a very human readable syntax to write sentences (like OK(Cell-1-1)), although we should understand that **the computer may represent it in another way**.

  E.g., it may use an array to hold all the strings X for which OK(X) holds, or use an array of rooms and store 1 into each cell with OK holds.
- The sentences can be interpreted as facts of the world. This is called the **semantics** (i.e., meaning) of the sentences.
- The semantics is **not known by the computer**. Instead, it is known only by the person writing the sentences.

## Inference

- Our primary objective: **given some facts that we know (or believe in), generate other facts that follows from the known facts**.
- In the computer: **given some sentences that are in our KB, generate other sentences entailed by those sentences.**

  So you can think that the term “entails by” means “follow from”, but only when we talk in terms of the knowledge base.



**Sentences** **Entails** **Sentence**

*Representation*

*World*

**Semantics** **Semantics**

**Facts** **Follows** **Fact**

Each KRL has its “way of making entailment”, or **inference procedure**.

An inference procedure is said to be **sound** if it only generate sentences that represent facts that are really followed by the known facts.

## Validity and Satisfiability

- A sentence is **valid**, or **necessarily true**, if it is true under all possible interpretations (semantics) of the sentence.

  E.g., “There is a stench at 1,1 or there is no stench at 1,1”.
- A sentence is **unsatisfiable** if it is false under all possible interpretations of the sentence—i.e., self-contradictory sentences.

  E.g., “There is a stench at 1,1 and there is no stench at 1,1”.
- A sentence is **satisfiable** if it is not unsatisfiable.

  E.g., “There is stench at 1,1”. It is false in **our** meaning of stench, but if “stench” means “agent”, it would be true.

Whether a sentence is satisfiable, unsatisfiable or valid **does not depend on the world**: it just depends on the sentences themselves.

This is the **basis of automatic inference**: how the computer can reason without knowing the semantics of the sentences.

## Inference in computers

- Suppose the agent needs to establish whether room 2,2 is OK.
- The reasoning program has **no knowledge about what is a wumpus, what is a pit, or what is OK, etc**.
- But it **has a knowledge base KB**, filled with knowledge like “if a room has no wumpus and has no pit, then the room is OK”, or “If room 2,1 contains no stench then room 2,2 has no wumpus.”.
- So the reasoning program will try to establish that “if KB, then room 2,2 is OK” is a **valid sentence**.

  Or perhaps “KB and room 2,2 is not OK” is an unsatisfiable sentence.
- If this succeed, then the reasoning agent will conclude that room 2,2 is OK—**without knowing what is meant by OK**!
- Note that **there are things which semantics are fixed** in the language: e.g., if, then, not, and, or, …

## Knowledge Representation Language

So what is the language in which we represent our knowledge?

Let’s see what we need:

- **Expressive**: can express all the knowledge we want the program to know.
- **Unambiguous**: can express accurate knowledge with no confusion.
- **Context independent**: a sentence must not have different meaning in different context.
- **Compositional**: the truth of a composite sentence must depend **only** on the truth of the sentences composing it.
- **Easy to reason with**: we want the knowledge to be easily retrievable and combined to form new knowledge.

## Example: Natural Language as KRL?

- Very **expressive**: it can represent all the things we know!
  Except when it gets complicated, only lawyers can read it.
- But **ambiguous**: Some sentence can be understood in two ways.
  "Small dogs and cats"—are the cats small?
  "I saw her duck"—is "her" possessive? Is "duck" a noun or a verb?
- **Context dependent**: Same sentence means different things in different situations.
  What is meant by "I" and "her" above?
- And not **compositional**: two phrases combine to mean completely unrelated things.
  Idioms, proverbs, …

Why these defects? **Natural language is designed for communication, not representation.**

Implication: understanding natural language is a separate task in AI.

## Logic as KRL

We will focus on using logic as our knowledge representation language:

- It is designed by logicians, to combat the downsides of natural languages.
- But we have to be more specific: **there are many logics**! They differ in what things in the world are represented, and in how much the agent believe in facts:

| **Logic** | **What is represented?** | **Fact is…** |
|---|---|---|
| **Propositional logic** | facts | true, false, unknown |
| **First-order logic** | facts, objects, relations | true, false, unknown |
| **Temporal logic** | facts, objects, relations, times | true, false, unknown |
| **Probability theory** | facts | true at some probability |
| **Fuzzy logic** | facts | true to some degree |

## Propositional Logic: a first, simple logic

- "Propositional logic" is a logic composed only of **propositions**.
- A proposition is an "atomic" sentence which **cannot be decomposed** into simpler ones. It can be either **true** or **false**.
- We will use **propositional symbols** to represent them. E.g., OK-1-1 might means cell 1,1 is okay, Breeze-2-1 might mean there is breeze at 2,1.
- There are two special symbols **True** and **False**, which always has value true and false respectively.
- A sentence can be a **simple sentence**, composed of **just a propositional symbol** or **special symbol**, and nothing else.
- A sentence can also be a **complex sentence**, composed of other propositional symbol(s) and **connectives**.

## All the connectives

If $P$ and $Q$ are sentences (either simple or complex), then

- $\neg P$ is a **negation** sentence. It has the intuitive meaning of logical "not".
- $P \wedge Q$ is a **conjunction** sentence. It has the intuitive meaning of logical "and". Logicians usually call $P \wedge Q$ a *conjunction* of the *conjuncts* $P$ and $Q$—in the same way that we call $1+2$ a sum.
- $P \vee Q$ is a **disjunction** sentence. It has the intuitive meaning of logical "or". $P$ and $Q$ are called the *disjuncts*.
- $P \Rightarrow Q$ is an **implication** sentence. If $P$ is false, then the implication is false. Otherwise, it has the same value as $Q$. $P$ is called the *premise*, $Q$ is called the *consequence*. (**Compositional!**)
- $P \Leftrightarrow Q$ is an **equivalence** sentence. It is true if $P$ and $Q$ are both true or both false.

The above list is from high to low precedence. Parentheses "()" can be used to override it.

## Validity and Truth tables

We can establish the validity of a sentence by listing the possible values of the related propositional symbols. E.g., if we need to know whether

$$((P \vee H) \wedge \neg H) \Rightarrow P$$

is a valid sentence, we can do the following:

| $P$ | $H$ | $P \vee H$ | $(P \vee H) \wedge \neg H$ | $((P \vee H) \wedge \neg H) \Rightarrow P$ |
|---|---|---|---|---|
| *False* | *False* | *False* | *False* | *True* |
| *False* | *True* | *True* | *False* | *True* |
| *True* | *False* | *True* | *True* | *True* |
| *True* | *True* | *True* | *False* | *True* |

The intermediate columns are not really needed: they are for easy reading only.

This can be used as an **inference procedure**: if KB contains $P \vee H$ and $\neg H$, and somebody query whether $P$ is entailed by KB, then we should answer "yes".

Still remember our earlier discussion about how computers deduce entailment?

## The need for a better inference procedure

- Using the Truth-table is a **sound inference procedure**: it will only tell you things that are *entailed* by KB.
  This is true regardless of the semantics of the propositions, and is true even if KB is self-contradictory.
- But there is a problem: **we have use all sentences in the knowledge base as premise** of our implication…
- and so we **have to consider all possible combinations of truth values of the propositional symbols!**
- If we have 100 symbols… we need to examine $2^{100}$ rows!!
- We should **not expect fast inference**: the problem to check for validity (or satisfiability) of even a very restricted expression is NP-complete.
- But we have seen in other AI problems that even if a problem is NP-complete, we can write a program that can *usually* solve it in reasonable time.

## Inference based on common rules

- Our strategy: use some common **inference rules** to do inference. Each time we focus on **just a few** sentences.
  This is possible because the other sentences of KB cannot invalidate our inference: a property known as **monotonicity**.
- We often write these rules like:
  $$\frac{\alpha, \beta, \ldots}{\gamma}$$
- or like $\alpha, \beta, \ldots \vdash \gamma$
- This means that **if we believe** $\alpha$**,** $\beta$**, …**, then it is **okay to believe in** $\gamma$
- For example, a reasoning system can repeatly **select some of the statements** in KB, apply one of these rules and add it back to KB, **until the queried result** is in KB (or nothing else can be derived).
- Now it becomes a **state-space search**—each states is a KB, and we look for a state in which the query is in KB.

## Common inference rules

- **Modus Ponens**:
  $\neg\alpha \vee \beta, \alpha \vdash \beta$
  (i.e., $\alpha \Rightarrow \beta, \alpha \vdash \beta$)
- **And-Elimination**:
  for any $1 \le i \le n$,
  $\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n \vdash \alpha_i$
- **And-Introduction**:
  $\alpha_1, \alpha_2, \ldots, \alpha_n \vdash \alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n$
- **Or-Introduction**:
  $\alpha_i \vdash \alpha_1 \vee \alpha_2 \vee \ldots \vee \alpha_n$
- **Double-Negation Elimination**:
  $\neg\neg\alpha \vdash \alpha$
- **Unit Resolution**:
  $\alpha \vee \beta, \neg\beta \vdash \alpha$
- **Resolution**:
  $\alpha \vee \beta, \neg\beta \vee \gamma \vdash \alpha \vee \gamma$
  (i.e., $\neg\alpha \Rightarrow \beta, \beta \Rightarrow \gamma \vdash \neg\alpha \Rightarrow \gamma$)

## Example inference

Suppose we have in KB

1. *Breeze-1-2* $\vee$ ($\neg$*Pit-1-1* $\wedge$ $\neg$*Pit-2-2* $\wedge$ $\neg$*Pit-1-3*)

2. $\neg$*Breeze-1-2*

Applying unit resolution to (1) and (2), we add into KB that

3. $\neg$*Pit-1-1* $\wedge$ $\neg$*Pit-2-2* $\wedge$ $\neg$*Pit-1-3*

Now apply and-elimination to (3):

4. $\neg$*Pit-2-2*

We can imagine that the system will generate $\neg$*Wumpus-2-2* later, and thus *OK-2-2* eventually using another rule, like

*Pit-2-2* $\vee$ *Wumpus-2-2* $\vee$ *OK-2-2*

## How good is such inference?

- **Inference is sound**: as long as the inference rules are correct, we always end up with entailed sentences.
- **Inference is complete** only if we restrict the number of forms of the possible sentences, or if we insert more rules to the system.
- **Worst case**: we can still require exponential time to find a proof.
- **What's good**: for most queries, only a few steps are required to arrive at the conclusion.
- **Special case**: if all sentences are of the form $P_1 \wedge P_2 \wedge \ldots \wedge P_n \Rightarrow Q$ ("**horn clauses**"), then there is a polynomial time inference procedure.
  We will discuss it later when we talk about first order logic.

## The downside

But there is a primary problem of propositional logic: it is very tedious.

- We need **many symbols and many, many sentences**.
- E.g., We can say *Pit-2-2* $\vee$ *Wumpus-2-2* $\vee$ *Okay-2-2*, but then we will have to repeat the thing for each of the cells.
- How to express that **there is only one wumpus**?
  *Wumpus-1-1* $\Rightarrow$ $\neg$*Wumpus-1-2*
  *Wumpus-1-1* $\Rightarrow$ $\neg$*Wumpus-1-3*
  …
  *Wumpus-4-4* $\Rightarrow$ $\neg$*Wumpus-4-3*
- But even then **many simple query can't be expressed**…
  How to express "where is the wumpus"?
- Next time we will **extend the language** a bit to support something like function calls, which makes things easier.