

# Heuristics and Logic

Searching smarter with a good guess, then writing knowledge in logic and checking what follows

Slides: Week 3 lecture · Seminar: Week 3 sheet · Code: [github.com/sakibanwar/comp5001-ai-code](https://github.com/sakibanwar/comp5001-ai-code), folder [week04-knowledge/logic](#) (and [maze4.txt](#) in [week02-search/maze](#) )

Sections 1 to 6 (heuristics) and 8 to 15 (logic) are adapted from the lecture notes of *CS50's Introduction to Artificial Intelligence with Python* by Brian Yu and David J. Malan (Harvard University), licensed under CC BY-NC-SA 4.0.

Sections 7 and 16 were written for COMP5001. The notes are shared under the same licence.

## THIS WEEK

By the end of the week you should be able to:

- explain what a heuristic is, and how greedy best-first search and A\* search use one;
- say what a statement is, and write sentences in propositional logic with not, and, or, if ... then and if and only if;
- use a truth table to work out whether a sentence is true in a model;
- explain entailment, and check it by model checking, by hand and with [logic.py](#).

**Remember Week 1?** You solved the knights and knaves puzzles by trying each possibility. Section 14 does the same by model checking, the way a computer would.

- |                                     |                                                                        |
|-------------------------------------|------------------------------------------------------------------------|
| 1. Where Week 2 left off            | 9. Symbols, and joining sentences: not, and, or                        |
| 2. A heuristic is a guess           | 10. If ... then, and if and only if                                    |
| 3. Greedy best-first search         | 11. Models and knowledge bases                                         |
| 4. Where greedy goes wrong          | 12. Entailment                                                         |
| 5. A* search                        | 13. Model checking                                                     |
| 6. A good guess never overestimates | 14. Knights and knaves, by model checking                              |
| 7. What is logic?                   | 15. In the code: <a href="#">logic.py</a> and <a href="#">harry.py</a> |
| 8. Knowledge-based agents           | 16. Heuristics and logic in the real world                             |
|                                     | 17. Check yourself, key terms and reading                              |

## 1 Where Week 2 left off

Last week, breadth-first search explored **29** squares of the Week 1 maze to find the 17-step path, and depth-first search explored 25. When you solved the same maze yourself in Week 1, you did not look at every square. You could **see where B was**, so you headed towards it.

DFS and BFS cannot do that. They try squares in a fixed order (newest first, or oldest first), whatever the maze looks like. This week's first half gives the search a sense of direction.

## UNINFORMED AND INFORMED SEARCH

**Uninformed search** uses no knowledge about where the goal is. DFS and BFS are uninformed.

**Informed search** uses extra knowledge about the problem, such as a guess at how far each state is from the goal, to look in the right direction first.

## 2 A heuristic is a guess

To head towards B, the search needs a number for each square that says roughly how far it is from B. In a maze, the simplest such number is the **Manhattan distance**: count how many rows and how many columns the square is from B, and add them up. It ignores the walls. (The name comes from walking between blocks in Manhattan, where you cannot cut diagonally across.)

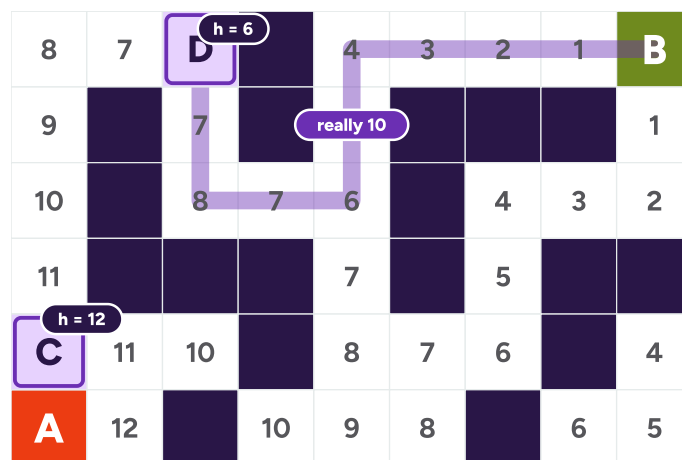


**Figure 1.** The Week 1 maze. The grey number in each square is its Manhattan distance to B. C and D are the two squares compared on the slide. The purple band is the real route from D to B.

### WORKED EXAMPLE

- **D** is in the top row, 6 columns left of B:  $0 + 6 = 6$ .
- **C** is 4 rows below B and 8 columns to the left:  $4 + 8 = 12$ .
- Ignoring walls, D looks much closer. But a wall stands between D and B, and the real route from D (the purple band) is **10** steps: down, along, and back up.

### HEURISTIC

A **heuristic  $h(n)$**  is a quick estimate of the cost from state **n** to the goal. In a maze we use the Manhattan distance.

### KEY IDEA

A heuristic is a **guess**. It is quick to work out and usually points the right way, but it can be wrong. Manhattan distance ignores walls, so it can be too low, but it is never too high (Section 6 explains why that matters).

### 3 Greedy best-first search

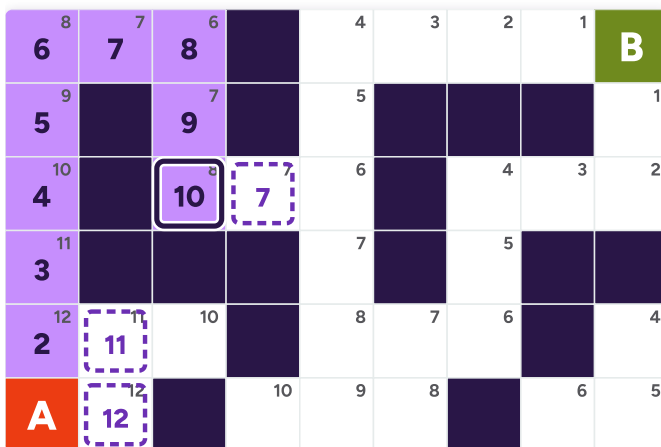
**Greedy best-first search** uses the loop from Week 2 and changes one line: which node comes out of the frontier. Instead of the newest (DFS) or the oldest (BFS), it takes the node with the **smallest  $h$** , the one that looks closest to the goal.

```
frontier = a node with the initial state
explored = an empty set
repeat:
    if the frontier is empty: there is no solution
    remove the node with the smallest  $h(n)$  from the frontier
    if the node is a goal: return the solution
    add the node to explored
    expand the node: add each neighbour to the frontier
        unless it is already in the frontier or explored
```

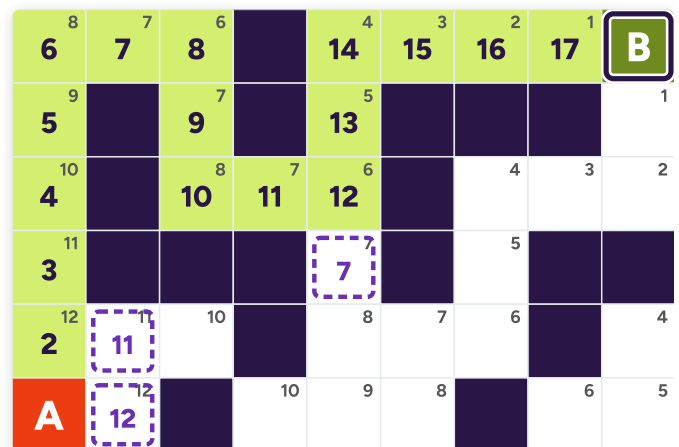
If two nodes have the same  $h$  (a **tie**), we take the one that joined the frontier first. Your own code may break ties differently, so your numbers can differ slightly.

#### Worked example: greedy on the Week 1 maze

The small number in the corner of each square is  $h$ . Lilac squares have been taken out of the frontier, numbered in order; dashed squares are waiting; the ringed square was taken out last.



**After step 10.** From A, up and right both had  $h = 12$ : a tie, so greedy took up (it joined the frontier first), and the same at step 3. It went up the left side and along the top to D (step 8,  $h = 6$ ). The wall blocks the way, so it goes down:  $h$  rises to 7, then 8. The only other squares waiting have  $h = 11$  and 12, so these still win.



**Finished (step 18).** Greedy explored only 18 squares (BFS: 29, DFS: 25) and found the 17-step path, the shortest. Every explored square is on the path.

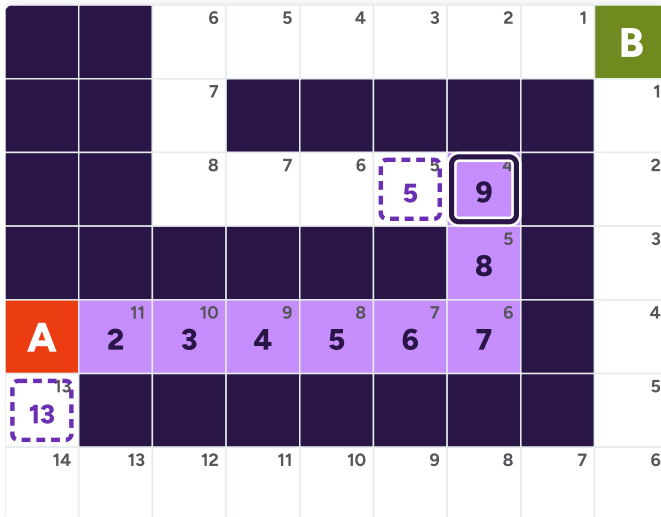
7 =  $h$ , the guess of steps to B      waiting in the frontier      explored (step number)      path

Here the guess was good, so greedy went almost straight to B. Is it always this good? No.

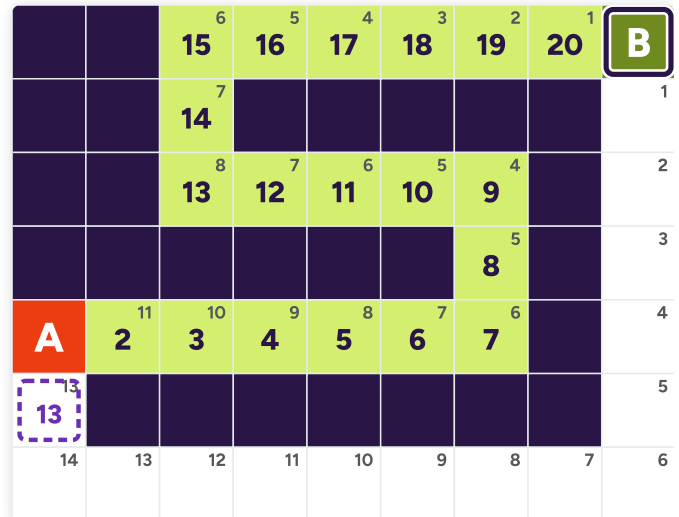
### 4 Where greedy goes wrong

The maze below is `maze4.txt` in `week02-search/maze`. Look at it before reading on: which way would you go from A? Most people go **down** and along the long bottom corridor, because they can see it leads straight

to the right-hand side.



**After step 9.** At step 2, right had  $h = 11$  and down had  $h = 13$ , so greedy went right. It reaches the corner with  $h = 4$ : B looks close, but a wall is in the way.



**Finished (step 21).** The route turned back to the left and  $h$  rose to 8, but the only other square waiting (below A) had  $h = 13$ , so greedy kept going. Its path is **20 steps**.

7 =  $h$ , the guess of steps to B    5 waiting in the frontier    9 explored (step number)    15 path

The shortest path goes down and along the bottom: **16 steps**. Greedy never found it, because at step 2 the bottom route looked worse (13 against 11), and after that the top route always looked better than the square below A.

### KEY IDEA

Greedy best-first search only looks **forward**, at the guess to go. It never counts the steps it has already taken, so a misleading guess can lead it a long way round.

- ✓ When the heuristic is good, greedy is often very fast.
- ✗ The path it finds may not be the shortest.
- ✗ A misleading heuristic can lead it away from a much shorter route.

## 5 A\* search

**A\* search** (said "A star") fixes greedy's mistake by adding the cost already paid. For each node it adds two numbers:

- $g(n)$ : the cost from the start to n, here the number of steps already taken. This is known exactly, because we walked it.
- $h(n)$ : the heuristic, the guessed cost from n to the goal.

### A\* SEARCH

Take the frontier node with the smallest  $f(n) = g(n) + h(n)$ : the cost so far plus the guessed cost to go. In other words, the estimated length of the whole path through n.

## Worked example: two squares in the frontier

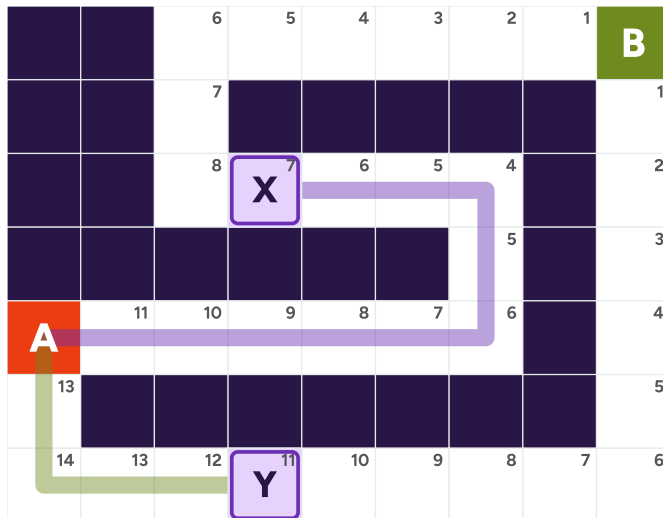


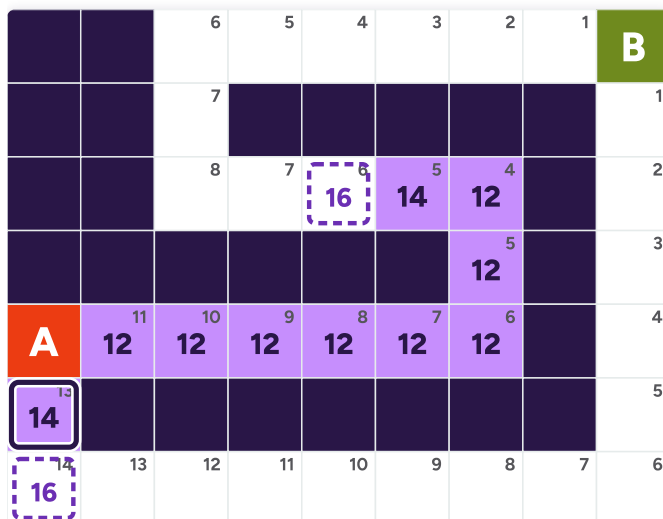
**Figure 2.** The trap maze. X (on the top route) and Y (on the bottom route) are both waiting in the frontier. The bands show how each was reached from A.

|                              | X  | Y  |
|------------------------------|----|----|
| $h(n)$ : guess to B          | 7  | 11 |
| $g(n)$ : steps from A so far | 11 | 5  |
| $f(n) = g(n) + h(n)$         | 18 | 16 |

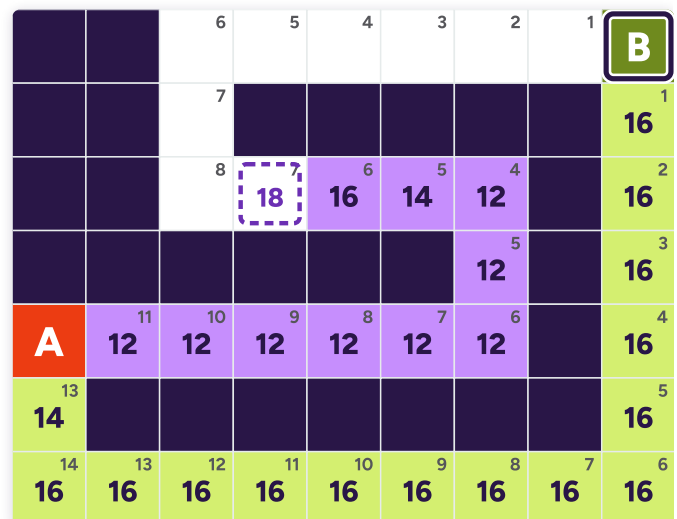
Greedy compares only  $h$  and takes **X** (7 is less than 11). A\* sees that X is already 11 steps from A, so a path through X is about 18 steps long, while a path through Y is about 16. It takes **Y**.

## Worked example: A\* on the trap maze

In these pictures the big number in each explored square is its  $f$ , not the step number. Ties are broken by taking the square nearer B (smaller  $h$ ).



**After step 11.** Along the top route  $f$  stays at 12: each step adds 1 to  $g$  and takes 1 off  $h$ . Round the corner  $f$  jumps to 14, so A\* also tries the square below A, which has  $f = 1 + 13 = 14$ .



**Finished (step 27).** The top route's next square has  $f = 18$ ; every square on the bottom route has  $f = 16$ . A\* follows the bottom route and finds the **16-step** path, the shortest.

7 =  $h$ , the guess of steps to B    [ ] waiting in the frontier    [ ] explored (number =  $f$ )    [ ] path

| Squares explored / path length | DFS      | BFS     | Greedy  | A*      |
|--------------------------------|----------|---------|---------|---------|
| Week 1 maze                    | 25 / 21  | 29 / 17 | 18 / 17 | 21 / 17 |
| maze2.txt                      | 194 / 30 | 77 / 30 | 54 / 30 | 50 / 30 |
| maze3.txt                      | 17 / 16  | 6 / 4   | 5 / 4   | 5 / 4   |
| maze4.txt (the trap)           | 21 / 20  | 32 / 16 | 21 / 20 | 27 / 16 |

Greedy often explores the fewest squares, but on the trap maze its path is 4 steps too long. BFS and A\* found the shortest path in every maze, and A\* always explored fewer squares than BFS, because the heuristic keeps it away from squares that lead the wrong way.

## Worked example: A\* with real costs

In Week 2, BFS on the road map returned the 55-minute route via Andover, because it counts roads, not minutes. A\* uses the minutes. Here  $g$  is the driving time so far, and  $h$  is a straight-line estimate of the time still to go (it can never be more than the real time, because no road is shorter than a straight line).

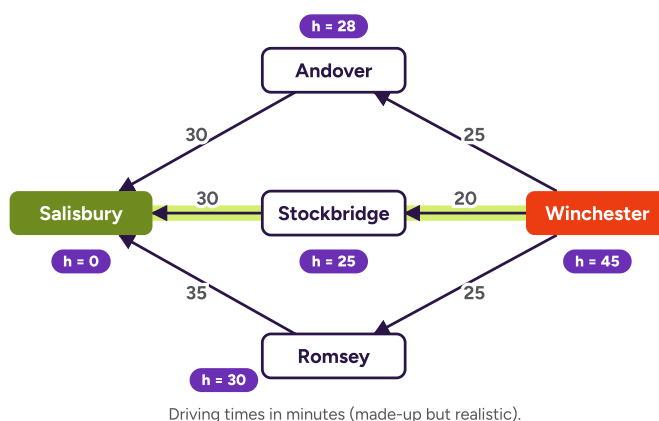


**Figure 3.** The Week 2 road map, with driving times in minutes on the roads and an estimate  $h$  for each town.

| Step | Take out                           | Add to the frontier ( $g + h =$ )                                           | Frontier after                        |
|------|------------------------------------|-----------------------------------------------------------------------------|---------------------------------------|
| 1    | <b>Winchester</b> ( $f = 0 + 45$ ) | Andover $25 + 28 = 53$ ; Stockbridge $20 + 25 = 45$ ; Romsey $25 + 30 = 55$ | Stockbridge 45, Andover 53, Romsey 55 |
| 2    | <b>Stockbridge</b> (45)            | Salisbury $50 + 0 = 50$                                                     | Salisbury 50, Andover 53, Romsey 55   |
| 3    | <b>Salisbury</b> (50)              | <b>goal ✓ via Stockbridge: 50 minutes</b>                                   |                                       |

A\* returns the optimal 50-minute route, and it never even expanded Andover or Romsey. Note that the goal test happens when a node is *taken out* of the frontier, not when it is added: that is what lets A\* compare every way of reaching the goal before it stops.

## 6 A good guess never overestimates

A straight line between two towns can never be longer than the road between them, because the road can bend but the straight line cannot. So a straight-line heuristic never guesses too high. Manhattan distance in a maze is the same: walls can only make the real route longer, never shorter.

## ADMISSIBLE HEURISTIC

A heuristic that **never overestimates** the true cost to the goal. A\* with an admissible heuristic always finds the shortest path.

## WHAT IF THE GUESS IS TOO HIGH?

Go back to Figure 2 and suppose the heuristic wrongly said that Y was 30 steps from B. Then  $f(Y) = 5 + 30 = 35$ . A\* would reach B along the top route ( $f = 20$ ) before it ever took Y, and it would return the 20-step path. An overestimate can make the best route look worse than it is, so A\* skips it.

## Other heuristics, and where they come from

Manhattan distance is the right guess for a maze, but every problem needs its own heuristic. Table 1 collects some common ones.

| Problem                               | Heuristic $h(n)$                                                                                          | Never overestimates?                                                                           |
|---------------------------------------|-----------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------|
| Maze: up, down, left, right           | Manhattan distance: rows apart + columns apart.                                                           | Yes: walls only make the route longer.                                                         |
| Maze where diagonal moves are allowed | The larger of rows apart and columns apart. A diagonal step covers one of each.                           | Yes. Manhattan distance would <i>not</i> be: it can be up to twice the real number of steps.   |
| Road map, sat-nav                     | Straight-line distance, as the crow flies. For a time estimate, the time to drive that line at top speed. | Yes: no road is shorter than a straight line.                                                  |
| Sliding-tile puzzle (the 8-puzzle)    | The number of tiles out of place. Better: the sum of each tile's Manhattan distance from its home square. | Yes: every tile out of place needs at least one move, in fact at least its Manhattan distance. |
| Any problem                           | $h = 0$ : no guess at all.                                                                                | Yes, but it gives no help: A* becomes BFS (question 4 in Check yourself).                      |
| Any problem                           | $2 \times$ Manhattan distance, or any guess that is inflated to speed things up.                          | No. A* may then return a longer path (seminar Activity 3).                                     |

**Table 1.** Some heuristics, and whether each one is admissible.

## RELAX THE PROBLEM

The reliable way to invent an admissible heuristic: **drop a rule** so that the problem becomes easier, then use the exact cost of the easier problem as the guess. With no walls, a maze costs exactly the Manhattan distance. If you could drive anywhere, a journey costs exactly the straight-line distance. If tiles could jump straight home, the 8-puzzle costs exactly the number of tiles out of place. An easier problem can never cost more than the real one, so the guess never overestimates.

Two more rules of thumb. A guess that is closer to the truth makes A\* explore fewer squares: on the 8-puzzle, the tiles-out-of-place guess and the Manhattan-distance guess are both admissible, but the second

is nearer the real answer and solves the puzzle far faster. And if you have two admissible heuristics, take the **larger** of the two at each node: it is still admissible, and at least as good as either on its own. Next week, planning uses a guess of the same kind: the number of goal facts that are not yet true.

*Going further:* CS50 and Russell & Norvig also describe **consistent** heuristics, which never drop by more than the cost of one step. Manhattan distance and straight-line distance are both consistent. You do not need this for COMP5001.

## 7 What is logic?

The second half of the week is about **logic**. In Week 1 you solved the knights and knaves puzzles by reasoning: “if A is a knight, then ...”. This half shows how to write that kind of knowledge down precisely, and how a computer can then work out what follows from it.

| Sentence                    | True or false?                                     |
|-----------------------------|----------------------------------------------------|
| Winchester is in Hampshire. | <b>true</b>                                        |
| $2 + 2 = 5$ .               | <b>false</b>                                       |
| Close the door!             | <i>neither: an order</i>                           |
| Is it raining?              | <i>neither: a question</i>                         |
| It is raining.              | <b>true</b> or <b>false</b> , depending on the day |

### STATEMENT AND LOGIC

A **statement** is a sentence that is either **true** or **false**. “ $2 + 2 = 5$ ” is a statement: it is false, but it is still true-or-false. Orders and questions are not statements.

**Logic** is a precise way to write statements, with rules for working out what follows from them. It is the language of the **symbolic AI** you met in Week 1.

Compare the last two rows. “Is it raining?” and “It is raining” use almost the same words, but only the second is a statement. Whether it is true depends on which world we are in: on a wet day it is true, on a dry day it is false. That idea, that the same sentence can be true in one world and false in another, is at the heart of this half of the week.

## 8 Knowledge-based agents

People reason from what they already know to reach new conclusions. CS50 starts with an example from Harry Potter. Suppose we know three things:

1. If it didn't rain, Harry visited Hagrid today.
2. Harry visited Hagrid or Dumbledore today, but not both.
3. Harry visited Dumbledore today.

Did it rain today? None of the three sentences says so, but we can work it out. Sentence 3 says Harry visited Dumbledore, and sentence 2 says he did not visit both, so:

4. Harry did not visit Hagrid today.

Now sentence 1 says that if it had not rained, Harry would have visited Hagrid. Sentence 4 says he did not. So:

5. It rained today.

### KNOWLEDGE-BASED AGENT, SENTENCE AND INFERENCE

A **knowledge-based agent** stores what it knows as sentences, and reasons with them to reach new conclusions.

A **sentence** is a statement written in a logic. CS50, the textbooks and the code all say "sentence", so from here on we do too.

**Inference** is working out new sentences from old ones, as we did for sentences 4 and 5.

We reached sentence 5 by being clever: we picked the right sentences in the right order. A computer needs a method that works without cleverness. The rest of the week builds one: first a precise way to write the sentences (Sections 9 and 10), then a way to check what follows (Sections 11 to 13).

## 9 Symbols, and joining sentences: not, and, or

### Symbols

Writing out "Harry visited Hagrid today" every time is slow, so we give each statement a short name:

It rained today. → **rain**

Harry visited Hagrid today. → **hagrid**

Harry visited Dumbledore today. → **dumbledore**

### PROPOSITIONAL SYMBOL

A short name for one statement. In any world, it is either **true** or **false**. Textbooks often use single capital letters: P, Q, R.

A symbol does not say whether its statement is true. It only names the statement. Whether **rain** is true depends on the world we are in.

Most sentences are built from smaller ones. Sentence 1, "If it didn't rain, Harry visited Hagrid today", contains two statements, "it rained" (with a "not") and "Harry visited Hagrid", joined by "if ... then". **Propositional logic** is logic built from symbols joined by five small words, called **connectives**: not, and, or, if ... then, and if and only if. This section covers the first three, and Section 10 the other two.

## Not, and, or

**not:** "It did not rain."  $\rightarrow$   **$\neg$ rain**

**and:** "Harry visited Hagrid and Dumbledore."  $\rightarrow$   **$\text{hagrid} \wedge \text{dumbledore}$**

**or:** "Harry visited Hagrid or Dumbledore."  $\rightarrow$   **$\text{hagrid} \vee \text{dumbledore}$**

Read  $\neg$  as "not",  $\wedge$  as "and" and  $\vee$  as "or". Each connective has a simple rule for when the whole sentence is true:

- **$\neg P$**  is true when P is false, and false when P is true: "not" flips the value.
- **$P \wedge Q$**  is true only when **both** P and Q are true.
- **$P \vee Q$**  is true when **at least one** of P and Q is true.

Here P and Q stand for any two sentences. A table that lists every combination of true and false, and the value of a sentence in each, is called a **truth table**.

| P     | Q     | $\neg P$ | $P \wedge Q$ | $P \vee Q$ |
|-------|-------|----------|--------------|------------|
| false | false | true     | false        | false      |
| false | true  | true     | false        | true       |
| true  | false | false    | false        | true       |
| true  | true  | false    | true         | true       |

**Table 2.** Truth table for not, and, or. Each row is one possible combination of values for P and Q.

### "OR" MEANS AT LEAST ONE

In logic,  $P \vee Q$  is true when P is true, or Q is true, **or both** (the last row of Table 2). Everyday English is not always like this: "for dessert you can have cookies or ice cream" usually means one but not both. That is called an *exclusive or*. To say "but not both" in logic, add  $\neg(P \wedge Q)$ .

### WORKED EXAMPLE: SENTENCE 2 IN SYMBOLS

"Harry visited Hagrid or Dumbledore today, but not both" has two parts. "Hagrid or Dumbledore" is  **$\text{hagrid} \vee \text{dumbledore}$** . "Not both" is  **$\neg(\text{hagrid} \wedge \text{dumbledore})$** . Both parts must be true, so join them with "and":

$$(\text{hagrid} \vee \text{dumbledore}) \wedge \neg(\text{hagrid} \wedge \text{dumbledore})$$

The brackets matter, as in arithmetic:  **$\neg(\text{hagrid} \wedge \text{dumbledore})$**  means "not both", while  **$\neg\text{hagrid} \wedge \text{dumbledore}$**  means "not Hagrid, and Dumbledore".

## 10 If ... then, and if and only if

**if ... then:** "If it is raining, then I am indoors."  $\rightarrow$   $\text{rain} \rightarrow \text{indoors}$

**if and only if:** "I am indoors if, and only if, it is raining."  $\rightarrow$   $\text{rain} \leftrightarrow \text{indoors}$

Read  $\text{rain} \rightarrow \text{indoors}$  as "rain implies indoors", and  $\text{rain} \leftrightarrow \text{indoors}$  as "rain if and only if indoors". CS50 calls these two connectives **implication** and **biconditional**.

### If ... then

Think of  $\text{rain} \rightarrow \text{indoors}$  as a promise: "If it is raining, I am indoors." When is the promise broken?

- It is raining and I am indoors: the promise is kept, so the sentence is **true**.
- It is raining and I am outside: the promise is broken, so the sentence is **false**. This is the **only** way to break it.
- It is not raining: the promise says nothing about where I am, so it cannot be broken, whether I am indoors or not. The sentence counts as **true**.

The last case surprises most people. If the "if" part is false, the whole "if ... then" sentence is true, whatever the "then" part says. Logically, we learn nothing from  $\text{rain} \rightarrow \text{indoors}$  on a dry day.

| P     | Q     | $P \rightarrow Q$ | $P \leftrightarrow Q$ |
|-------|-------|-------------------|-----------------------|
| false | false | true              | true                  |
| false | true  | true              | false                 |
| true  | false | false             | false                 |
| true  | true  | true              | true                  |

**Table 3.** Truth table for if ... then ( $\rightarrow$ ) and if and only if ( $\leftrightarrow$ ).

### If and only if

$P \leftrightarrow Q$  is an "if ... then" in both directions:  $P \rightarrow Q$  **and**  $Q \rightarrow P$ . "I am indoors if and only if it is raining" says that rain means indoors, and also that indoors means rain. So on a dry day I am outside.  $P \leftrightarrow Q$  is **true** exactly when P and Q have the **same** value.

#### KEY IDEA

$P \rightarrow Q$  is false only when P is true and Q is false.  $P \leftrightarrow Q$  is true when P and Q are the same.

Table 4 collects all five connectives.

| Connective                     | Symbol            | Example               | True when                   |
|--------------------------------|-------------------|-----------------------|-----------------------------|
| not                            | $\neg$            | $\neg P$              | P is false                  |
| and                            | $\wedge$          | $P \wedge Q$          | both are true               |
| or                             | $\vee$            | $P \vee Q$            | at least one is true        |
| if ... then (implication)      | $\rightarrow$     | $P \rightarrow Q$     | P is false, or Q is true    |
| if and only if (biconditional) | $\leftrightarrow$ | $P \leftrightarrow Q$ | P and Q have the same value |

**Table 4.** The five connectives of propositional logic.

### WORKED EXAMPLE: THE HARRY KNOWLEDGE IN SYMBOLS

Sentence 1, "If it didn't rain, Harry visited Hagrid today", becomes  $\neg \text{rain} \rightarrow \text{hagrid}$ . With sentence 2 from Section 9 split into its two parts, everything we know is:

1.  $\neg \text{rain} \rightarrow \text{hagrid}$
2.  $\text{hagrid} \vee \text{dumbledore}$
3.  $\neg(\text{hagrid} \wedge \text{dumbledore})$
4.  $\text{dumbledore}$

These four sentences are the knowledge base used in Section 13 and in `harry.py`.

## 11 Models and knowledge bases

Whether a sentence is true depends on the world. With two symbols, `rain` and `hagrid`, there are four possible worlds: both true, only `rain` true, only `hagrid` true, or both false.

### MODEL

One possible world: a **true** or **false** value for every symbol. For example,  $\{\text{rain} = \text{true}, \text{hagrid} = \text{false}\}$  is a model: it rained, and Harry did not visit Hagrid.

Each symbol can be true or false, so every new symbol **doubles** the number of possible worlds. With  $n$  symbols there are  $2 \times 2 \times \dots \times 2 = 2^n$  models.

| Symbols | Models (worlds) |
|---------|-----------------|
| 2       | 4               |
| 3       | 8               |
| 4       | 16              |
| 10      | 1,024           |
| 20      | about 1 million |
| 30      | about 1 billion |

**Table 5.** The number of models doubles with every symbol.

## Is a sentence true in a model?

To find out, put the model's values into the sentence and work from the inside out, using the rules in Table 4.

### WORKED EXAMPLE: ONE SENTENCE IN ONE MODEL

Is  $\neg \text{rain} \rightarrow \text{hagrid}$  true in the model  $\{\text{rain} = \text{false}, \text{hagrid} = \text{false}\}$ ?

1.  $\text{rain}$  is false, so  $\neg \text{rain}$  is **true**.
2.  $\text{hagrid}$  is **false**.
3.  $\text{true} \rightarrow \text{false}$  is **false** (the third row of Table 3).

So the sentence is false in this model: on a dry day when Harry did not visit Hagrid, sentence 1 is broken.

### KNOWLEDGE BASE (KB)

The set of sentences an agent knows to be **true**. The KB is true in a model only when **every** sentence in it is true, so it behaves like one big sentence joined by  $\wedge$ .

Knowledge rules worlds out. Suppose the KB is a single sentence,  $\text{rain} \vee \text{hagrid}$ . It is false only in the world where both symbols are false, so that world is ruled out, and three are left. Every sentence we add can only rule out more worlds, never bring one back.

## 12 Entailment

If "It is a Tuesday in January" is true, must "It is January" be true? Yes: every world in which it is a Tuesday in January is a world in which it is January. The other way round fails: on a Wednesday in January, "It is January" is true but "It is a Tuesday in January" is not. One world like that is enough to show that the second sentence does not follow from the first.

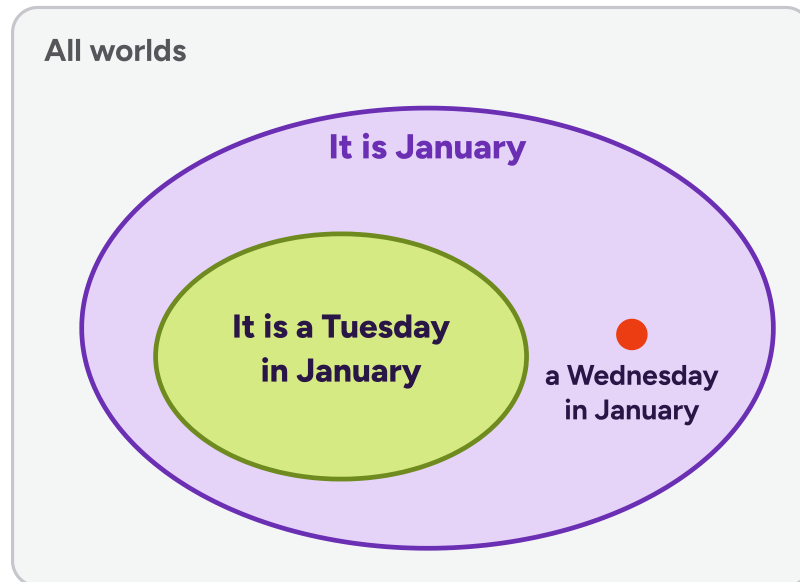


**Figure 4.** Every world where it is a Tuesday in January (green) is inside the worlds where it is January (purple). The red dot is a world where “It is January” is true but “It is a Tuesday in January” is false.

### ENTAILMENT

A **entails** B when B is **true** in **every** world where A is **true**. We also say that B **follows from** A. It is written  $A \models B$ , read “A entails B”.

For a knowledge-based agent the question is always the same: does the knowledge base entail the sentence we want to know about, the **query**? In symbols,  $KB \models \text{query}$ ? For the Harry example: does our knowledge base entail **rain**?

### ENTAILMENT IS NOT THE SAME AS “IF ... THEN”

The arrow  $\rightarrow$  is a connective: it is a word *inside* a sentence, and the sentence it makes can be true in some worlds and false in others. Entailment  $\models$  is not part of any sentence. It is a claim *about* sentences: in every world where the first is true, the second is true too.

## 13 Model checking

Entailment says “B is true in **every** world where the KB is true”. The most direct way to check that is to look at every world. That is **model checking**.

```
to check whether  $KB \models \text{query}$ :
  for each model (every true/false choice of the symbols):
    if the KB is true in this model:
      if the query is false in this model: return “no”
  return “yes”
```

Models where the KB is false are skipped: they are worlds that contradict what we know, so they do not matter. If the query is true in every model that is left, the KB entails it.

### WORKED EXAMPLE: DID IT RAIN?

The knowledge base is the four sentences at the end of Section 10, and the query is **rain**. There are three symbols, so  $2^3 = 8$  models (Table 6). Take the sentences one at a time and rule out the models where each one is false:

- Sentence 4, **dumbledore**, is false in the four models where **dumbledore** is false. Rule them out.
- Sentence 3, **¬(hagrid ∧ dumbledore)**, is false where both are true. Two more models go.
- Sentence 1, **¬rain → hagrid**, is false where **rain** and **hagrid** are both false (the worked example in Section 11). One more model goes.
- Sentence 2, **hagrid ∨ dumbledore**, rules out nothing new: the other sentences have already removed every model where it is false.

One model is left: **rain** true, **hagrid** false, **dumbledore** true. **rain** is true in it, so it is true in every model where the KB is true: **KB ⊨ rain**. It rained, the same answer as in Section 8, but this time found without any cleverness.

| rain  | hagrid | dumbledore | KB true?                          |
|-------|--------|------------|-----------------------------------|
| false | false  | false      | no: breaks 4                      |
| false | false  | true       | no: breaks 1 (no rain, so Hagrid) |
| false | true   | false      | no: breaks 4                      |
| false | true   | true       | no: breaks 3 (not both)           |
| true  | false  | false      | no: breaks 4                      |
| true  | false  | true       | yes                               |
| true  | true   | false      | no: breaks 4                      |
| true  | true   | true       | no: breaks 3 (not both)           |

**Table 6.** Model checking the Harry knowledge base. Crossed-out models break at least one sentence; the green model is the only one where the KB is true.

The same table answers other queries too. Does the KB entail **¬hagrid**? In the one model left, **hagrid** is false, so yes: Harry did not visit Hagrid. Does the KB entail **hagrid**? No: the one model left is a counter-example.

### THE CATCH

Model checking always gives the right answer, but it looks at every model, and the number of models doubles with every symbol (Table 5). Three symbols is easy. A real knowledge base with hundreds of symbols has more models than there are atoms in the universe. Next week starts here: ways to reason that do not check every world.

## 14 Knights and knaves, by model checking

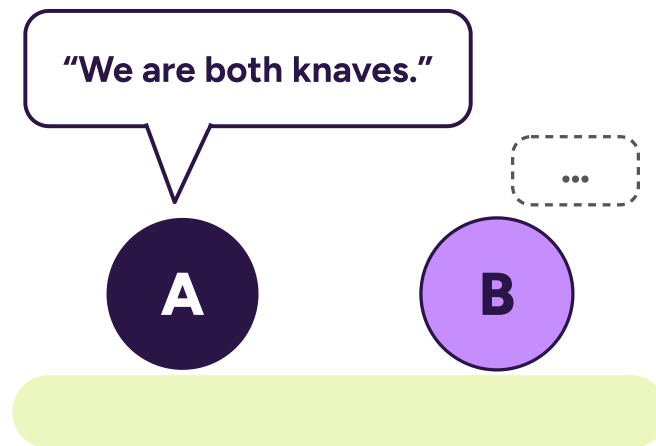


**Figure 5.** The Week 1 puzzle: every person is a knight (always tells the truth) or a knave (always lies). A says "We are both knaves." B says nothing.

In Week 1 you solved this puzzle in words. Most people reason like this: "Suppose A is a knight. Then what A said is true, so A is a knave. But A cannot be both, so A is not a knight. So A is a knave, which means A was lying: they are *not* both knaves. A is a knave, so B must be the one who is not. B is a knight." That is good reasoning, but it needs a clever idea ("suppose A is a knight") and a different clever idea for every new puzzle. A computer needs a method with no cleverness in it. This section builds one in four steps, and the steps work for any knights-and-knaves puzzle.

### Step 1: name the statements

A statement is anything that is true or false. Here there are four, one for each thing a person might be. Give each a short name, a propositional symbol:

- **AKnight**: A is a knight.
- **AKnave**: A is a knave.
- **BKnight**: B is a knight.
- **BKnave**: B is a knave.

A symbol does not say whether its statement is true. It just names it. In one world **AKnight** is true; in another it is false. Our job is to find the world that fits everything we know.

### Step 2: write down the rules of the island

The puzzle says every person is a knight or a knave. We also know nobody is both. In plain words, for A: "A is a knight or a knave, and not both." In symbols:

$$(\text{AKnight} \vee \text{AKnave}) \wedge \neg(\text{AKnight} \wedge \text{AKnave})$$

Read it as: (A is a knight or A is a knave) *and* it is *not* the case that (A is a knight *and* A is a knave). The first bracket rules out "neither"; the second rules out "both". Write the same sentence for B. Together these are **sentence 1** of the knowledge base.

### Step 3: write down what A said

A said: "We are both knaves." As a sentence of logic that is simply  $AKnave \wedge BKnave$ . But we cannot add it to the knowledge base as it stands, because we do not know whether A was telling the truth. What we do know is the rule about who tells the truth:

- If A is a knight, then what A said is true. In symbols, **sentence 2:**  $AKnight \rightarrow (AKnave \wedge BKnave)$
- If A is a knave, then what A said is false. **Sentence 3:**  $AKnave \rightarrow \neg(AKnave \wedge BKnave)$

This is the one trick in the whole method: **never write down a person's statement on its own. Write "if knight, then it is true" and "if knave, then it is false"**. The second sentence uses a "not" in front of the statement:  $\neg(\text{both knaves})$  means "at least one of them is not a knave". B said nothing, so there is nothing to write for B.

Those three sentences are the whole knowledge base. Nothing in it says what A or B is; the answer has to follow from it.

### Step 4: check every world

Four symbols give  $2^4 = 16$  worlds, but sentence 1 throws out every world where someone is both or neither. Only four worlds survive it: each of A and B is a knight or a knave. Now test sentences 2 and 3 in each of the four. The question to ask in each world is: *is what A said true here, and is that what A's kind allows?*

| A is a | B is a | "Both knaves" is | Sentence 2 (knight $\rightarrow$ true)         | Sentence 3 (knave $\rightarrow$ false)       | Fits? |
|--------|--------|------------------|------------------------------------------------|----------------------------------------------|-------|
| knight | knight | false            | broken: A is a knight but said something false | holds: A is not a knave, so it says nothing  | no    |
| knight | knave  | false            | broken: the same                               | holds                                        | no    |
| knave  | knight | false            | holds: A is not a knight, so it says nothing   | holds: A is a knave and said something false | yes   |
| knave  | knave  | true             | holds                                          | broken: A is a knave but said something true | no    |

**Table 7.** The four worlds left after sentence 1, tested against sentences 2 and 3. Only one fits the whole knowledge base.

Two things in the table deserve a second look. First, an "if ... then" sentence *holds* whenever its "if" part is false: in the third row A is not a knight, so sentence 2 is not tested and cannot be broken. That is the "promise not tested" rule from Section 10, and it is what lets one sentence sit quietly while the other does the work. Second, the last row is the one people find surprising: two knaves looks consistent with "we are both knaves", until you remember that a knave cannot say something true.

### Reading off the answer

Only one world fits: A is a knave and B is a knight. Every sentence we care about is true in every surviving world (there is only one), so the knowledge base entails  $AKnave$  and it entails  $BKnight$ :  $KB \models AKnave \wedge BKnight$ . This is the same answer as Week 1, found with no cleverness: name the statements, write the rules, check every world. If two worlds had survived, the puzzle would have two answers; if none had, the puzzle would contradict itself.

## RECIPE FOR ANY KNIGHTS-AND-KNAVES PUZZLE

1. Two symbols per person, `XKnight` and `XKnav`.
2. For each person, the island rule: one kind, not both.
3. For each statement  $S$  that a person  $X$  makes, two sentences:  $XKnight \rightarrow S$  and  $XKnav \rightarrow \neg S$ .
4. Check every world and keep the ones where every sentence is true. The seminar sheet uses the recipe on Week 1's harder puzzle, where  $B$  speaks too, and on a new one; CS50's *Knights* project writes the same puzzles in `logic.py`, which is next.

## 15 In the code: `logic.py` and `harry.py`

The practical uses CS50's `logic.py` (folder `week04-knowledge/logic` in the module code). It gives a Python class for each part of propositional logic: `Symbol`, `Not`, `And`, `Or`, `Implication` and `Biconditional`, plus the function `model_check`. Here is `harry.py`, the Harry example:

```
from logic import *

rain = Symbol("rain")
hagrid = Symbol("hagrid")
dumbledore = Symbol("dumbledore")

knowledge = And(
    Implication(Not(rain), hagrid),
    Or(hagrid, dumbledore),
    Not(And(hagrid, dumbledore)),
    dumbledore
)

print(model_check(knowledge, rain))
```

The four lines inside `And(...)` are the four sentences from Section 10, in the same order. `And` joins them into one knowledge base, true only when all four are. The last line asks whether the knowledge base entails `rain`, and prints **True**.

### How `model_check` works

`model_check` does exactly what Section 13 describes, but it builds the models one symbol at a time. Its helper, `check_all`, takes the list of symbols that have no value yet:

- If every symbol has a value, the model is complete. If the knowledge base is true in it, return whether the query is true; if not, the model does not matter, so return false.
- Otherwise, take one symbol without a value, make two copies of the model (one where the symbol is true, one where it is false), and check both. The answer is true only if both checks say true.

A function that calls itself like this is called **recursive**. Each call fixes one more symbol, so with three symbols it reaches all 8 complete models, the rows of Table 6. Each sentence class has an `evaluate(model)` method that works out whether the sentence is true in a model, using the rules in Table 4.

### TRY IT IN THE PRACTICAL

- Run `python harry.py`. Then change the query to `hagrid`, and to `Not(hagrid)`, and predict the answer before you run it.
- Delete `dumbledore` from the knowledge base. Does it still entail `rain`? Use Table 6 to explain why not.
- Write the knights and knaves puzzle from Section 14 with `logic.py` and ask `model_check` about each of the four symbols.
- Stretch: add greedy best-first search or A\* to `maze.py` (in `week02-search`) and try it on `maze4.txt`.

## 16 Heuristics and logic in the real world

- **Sat-navs and video games:** route finders use A\* (with many refinements) and a heuristic such as straight-line distance.
- **Computer chips and software:** model checking is used to show that a design can never reach a bad state, by checking every state it could be in. Edmund Clarke, Allen Emerson and Joseph Sifakis shared the 2007 Turing Award for it. Industrial model checkers use far cleverer methods than a truth table, but the idea is the same: try every possibility.
- **Railways:** logic is used to check that signalling systems can never send two trains onto the same stretch of track.
- **Product options:** when you build a car or a computer online, logic works out which options can go together.

A question to think about: a railway signalling system has been checked by model checking, and in every world that fits its knowledge base no two trains share a track. Can we trust it completely? The check is only as good as the knowledge base. A missing or wrong sentence, or something the symbols do not cover (a broken sensor, a driver who ignores a signal), is outside the proof. People who understand the railway must check the sentences, and the system still needs testing in the real world.

## Check yourself

Try these without looking back. The answers are at the end.

1. In Figure 1, what is  $h$  for the square directly below D? Is the real number of steps from that square to B more or less than  $h$ ?
2. The frontier holds three squares with  $h = 9, 4$  and  $6$ , and  $g = 2, 10$  and  $6$ . Which one does greedy best-first search take next? Which one does A\* take?
3. In `maze4.txt`, why does greedy best-first search find a 20-step path when a 16-step path exists?
4. Suppose  $h = 0$  for every square. Does it ever overestimate? What would A\* do then?
5. A robot in a grid can also move diagonally, one square at a time. Is Manhattan distance still admissible? Suggest a heuristic that is.

6. Which of these are statements? "Winchester is in Scotland." "Shut the window!" "Is the robot holding a block?"
7. Let P be "It is sunny" and Q be "We go to the beach". Write in symbols: (a) "It is not sunny." (b) "If it is sunny, we go to the beach." (c) "We go to the beach if and only if it is sunny."
8. P is false and Q is true. Which of these are true?  $P \wedge Q$ ,  $P \vee Q$ ,  $P \rightarrow Q$ ,  $Q \rightarrow P$ ,  $P \leftrightarrow Q$ .
9. How many models are there with 4 symbols? With 10?
10. Does **rain v hagrid** entail **rain**? Give a model that shows why or why not.
11. CS50's example. P: "It is a Tuesday." Q: "It is raining." R: "Harry will go for a run." The KB is  $(P \wedge \neg Q) \rightarrow R$ , P and  $\neg Q$ . Use model checking to decide whether  $KB \models R$ .
12. The first Week 1 knights and knaves puzzle: A says "I am both a knight and a knave." Using the symbols **AKnight** and **AKnave**, write the knowledge base and check the four models. What is A?

## Key terms

|                                     |                                                                                                                       |
|-------------------------------------|-----------------------------------------------------------------------------------------------------------------------|
| <b>Uninformed / informed search</b> | Uses no knowledge of where the goal is (DFS, BFS) / uses a heuristic to look in the right direction.                  |
| <b>Heuristic, <math>h(n)</math></b> | A quick estimate of the cost from <b>n</b> to the goal.                                                               |
| <b>Manhattan distance</b>           | Rows apart + columns apart, ignoring walls.                                                                           |
| <b>Greedy best-first search</b>     | Takes the frontier node with the smallest <b>h</b> ; fast, but the path may not be the shortest.                      |
| <b><math>g(n)</math></b>            | The cost from the start to <b>n</b> (steps so far).                                                                   |
| <b>A* search</b>                    | Takes the frontier node with the smallest <b><math>g(n) + h(n)</math></b> .                                           |
| <b>Admissible heuristic</b>         | Never overestimates; with it, A* finds the shortest path.                                                             |
| <b>Relaxed problem</b>              | The problem with a rule dropped. Its exact cost is an admissible heuristic for the real problem.                      |
| <b>Statement</b>                    | A sentence that is either true or false.                                                                              |
| <b>Logic</b>                        | A precise way to write statements, with rules for working out what follows.                                           |
| <b>Knowledge-based agent</b>        | An agent that stores what it knows as sentences and reasons with them.                                                |
| <b>Sentence</b>                     | A statement written in a logic.                                                                                       |
| <b>Inference</b>                    | Working out new sentences from old ones.                                                                              |
| <b>Propositional symbol</b>         | A short name for one statement, such as <b>rain</b> or P.                                                             |
| <b>Connectives</b>                  | not ( $\neg$ ), and ( $\wedge$ ), or ( $\vee$ ), if ... then ( $\rightarrow$ ), if and only if ( $\leftrightarrow$ ). |
| <b>Truth table</b>                  | A table of every combination of true and false, with the value of a sentence in each.                                 |
| <b>Model</b>                        | One possible world: a true or false value for every symbol. $n$ symbols give $2^n$ models.                            |
| <b>Knowledge base (KB)</b>          | The sentences an agent knows to be true.                                                                              |

|                                             |                                                               |
|---------------------------------------------|---------------------------------------------------------------|
| <b>Entailment, <math>A \models B</math></b> | B is true in every model where A is true: B follows from A.   |
| <b>Query</b>                                | The sentence we want to know about: does the KB entail it?    |
| <b>Model checking</b>                       | Checking $KB \models \text{query}$ by looking at every model. |

## Reading and next week

- Russell & Norvig, *Artificial Intelligence: A Modern Approach* (4th ed.): sections 3.5 and 3.6 (informed search and heuristics), and sections 7.1 to 7.4 (logical agents and propositional logic; 7.2, the wumpus world, is a longer example you can skim).
- Poole & Mackworth, *Artificial Intelligence: Foundations of Computational Agents* (3rd ed., free at [artint.info](http://artint.info)): section 3.6 and section 5.1.

**Next week: Knowledge and Planning.** Model checking doubles its work with every symbol. Next week starts with ways to reason that do not check every world (inference rules, and a first look at first-order logic), then uses facts and actions to **plan**, and solves the Week 1 river crossing.

## Answers to Check yourself

1. The square below D is 1 row and 6 columns from B, so  $h = 7$ . The real route goes down, along and back up round the wall: **9** steps, which is more than  $h$ . Manhattan distance can be too low, never too high.
2. Greedy takes the square with  $h = 4$  (the smallest  $h$ ).  $A^*$  works out  $f = g + h$ :  $2 + 9 = 11$ ,  $10 + 4 = 14$  and  $6 + 6 = 12$ , so it takes the square with  $h = 9$  ( $f = 11$ ). The square that looks closest has already cost 10 steps to reach.
3. At step 2 the square to the right of A has  $h = 11$  and the square below has  $h = 13$ , so greedy goes right. From then on, every square on the top route has a smaller  $h$  than the square below A, so greedy never goes back. It only looks at  $h$  and never counts the steps already taken.
4. No: 0 is never more than the real distance, so  $h = 0$  is admissible, and  $A^*$  still finds the shortest path. But  $f = g + 0$ , so  $A^*$  just takes the square with the fewest steps so far. In a maze that is the same as BFS: no sense of direction at all.
5. **No.** A diagonal step moves one row and one column at once. A square 3 rows and 3 columns away is 3 steps, but Manhattan distance says 6: an overestimate, so  $A^*$  might miss the shortest path. Use the **larger** of rows apart and columns apart (3 here). It never overestimates, because every step changes a row or a column by at most one.
6. "Winchester is in Scotland" is a statement (a **false** one). "Shut the window!" is an order and "Is the robot holding a block?" is a question, so neither is a statement.
7. (a)  $\neg P$  (b)  $P \rightarrow Q$  (c)  $Q \leftrightarrow P$ , which is the same as  $P \leftrightarrow Q$ .
8.  $P \wedge Q$  is **false** ( $P$  is false).  $P \vee Q$  is **true** ( $Q$  is true).  $P \rightarrow Q$  is **true** (the "if" part is false).  $Q \rightarrow P$  is **false** ( $Q$  is true and  $P$  is false: the only way to break an "if ... then").  $P \leftrightarrow Q$  is **false** (they differ).
9.  $2^4 = 16$ , and  $2^{10} = 1,024$ .

10. **No.** In the model  $\{\text{rain} = \text{false}, \text{hagrid} = \text{true}\}$ ,  $\text{rain} \vee \text{hagrid}$  is true but  $\text{rain}$  is false. One such model is enough to show that  $\text{rain}$  does not follow.
11. There are 8 models. P must be true, which rules out the 4 models where P is false. Q must be false, which rules out 2 more. That leaves P true, Q false, with R true or R false. With P true and Q false,  $P \wedge \neg Q$  is true, so  $(P \wedge \neg Q) \rightarrow R$  is false when R is false: that model goes too. One model is left, and R is true in it, so **KB**  
 $\models \mathbf{R}$ : Harry will go for a run.
12. KB:  $(\text{AKnight} \vee \text{AKnave}) \wedge \neg(\text{AKnight} \wedge \text{AKnave})$ ,  $\text{AKnight} \rightarrow (\text{AKnight} \wedge \text{AKnave})$  and  $\text{AKnave} \rightarrow \neg(\text{AKnight} \wedge \text{AKnave})$ . Both true: breaks the first sentence. Both false: breaks the first sentence. Knight only: the second sentence needs A to be a knave as well, so it is false. Knave only: the third sentence is  $\neg(\text{false} \wedge \text{true}) = \text{true}$ , so this model fits. **A is a knave.**

---

COMP5001 Artificial Intelligence, University of Winchester, 2026/27 · Dr Sakib Anwar ·

Sakib.Anwar@winchester.ac.uk

Sections 1 to 6 and 8 to 15 are adapted from the lecture notes of CS50's *Introduction to Artificial Intelligence with Python* by Brian Yu and David J. Malan, Harvard University, licensed under CC BY-NC-SA 4.0. Shared under the same licence.