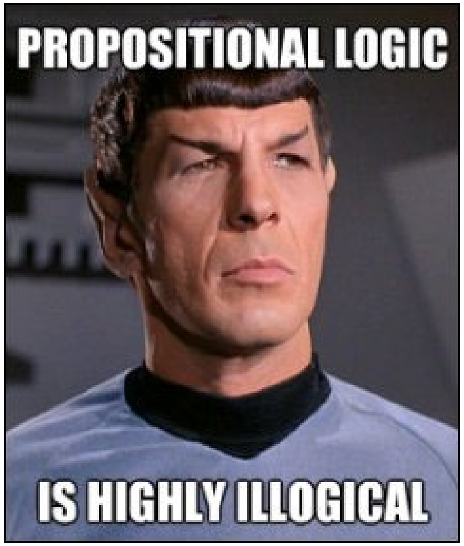
Logic 1
Propositional Logic
DV2557

Dr. Prashant Goswami
Assistant Professor, BTH (DIDA)
prashantgos.github.io/

prashant.goswami@bth.se



First, let us define a testbed…

Meet the Wumpus!

The Wumpus World

- A cave consisting of rooms connected vertically and horizontally.
- Somewhere in the cave lurks the Wumpus.
- The Wumpus can be killed by the player, but the player only has one arrow.
- Some rooms have bottomless pits.
- Goal is to find the gold treasure!
- Wumpus world is a well-known testbed for logic, first is from 1972.

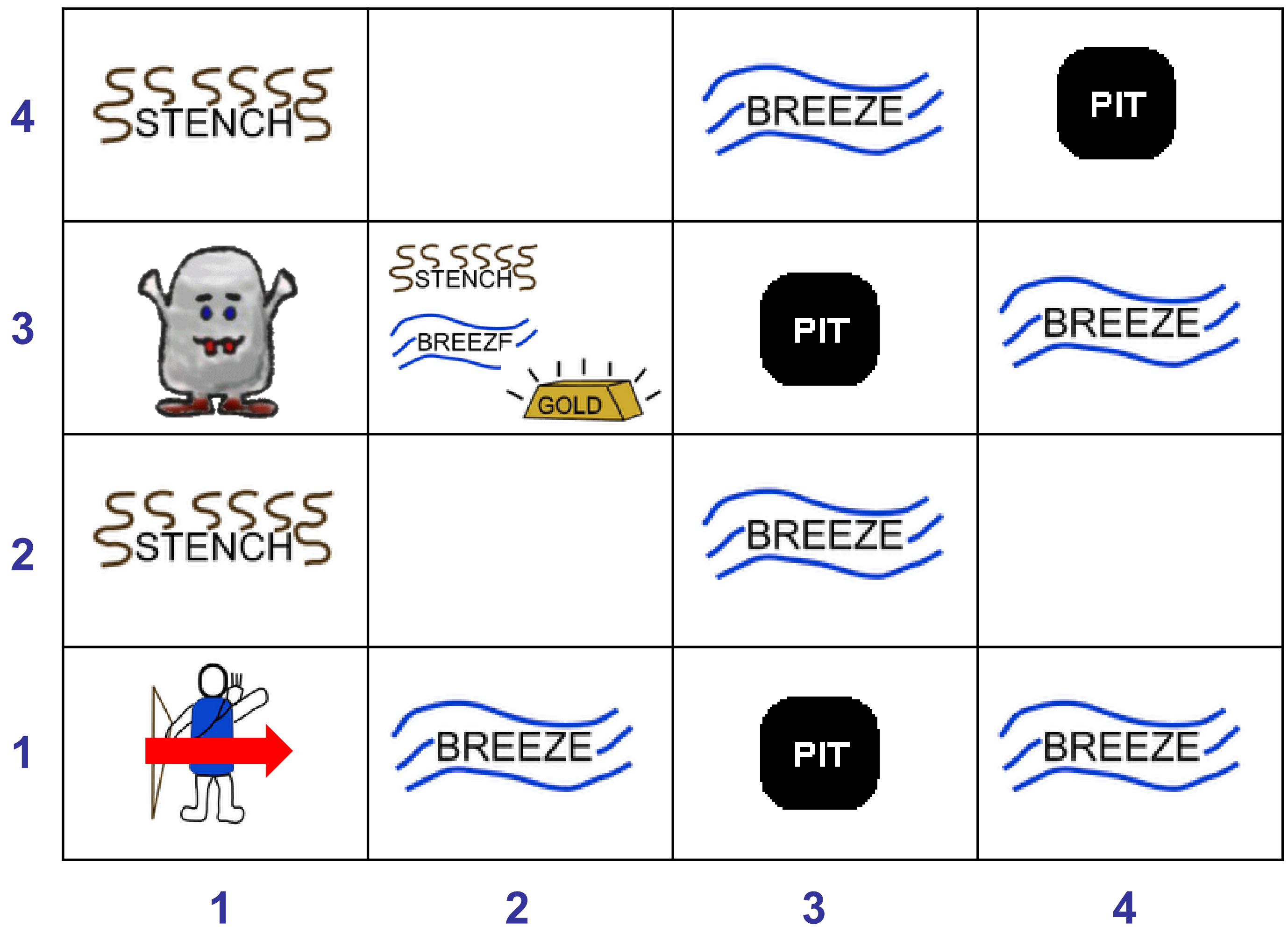
The Wumpus World

- Score:
  - +1000 for picking up the gold.
  - -1000 for falling into a pit or getting eaten by the Wumpus.
  - -1 for each action taken.
  - -10 for shooting the arrow.
- Environment:
  - 4x4 grid in our example.
  - Player starts at (1,1), facing right.
  - Randomly placed pits, Wumpus and gold.
- Actions:
  - Turn 90^0 left or right
  - Move forward
  - Shoot
  - Grab

The Wumpus World

- Sensors:
  - In squares next to the Wumpus the player perceives a stench (not diagonally).
  - In the squares next to a pit the player perceives a breeze (not diagonally).
  - In the square with the gold treasure, the player perceives a glitter.
  - If the player walks into a wall, it perceives a bump.
  - If the Wumpus is killed, a scream is heard all over the cave.
  - Percepts: [Stench, Breeze, Glitter, Bump, Scream]
  - Example: [Stench, Breeze, None, None, None]

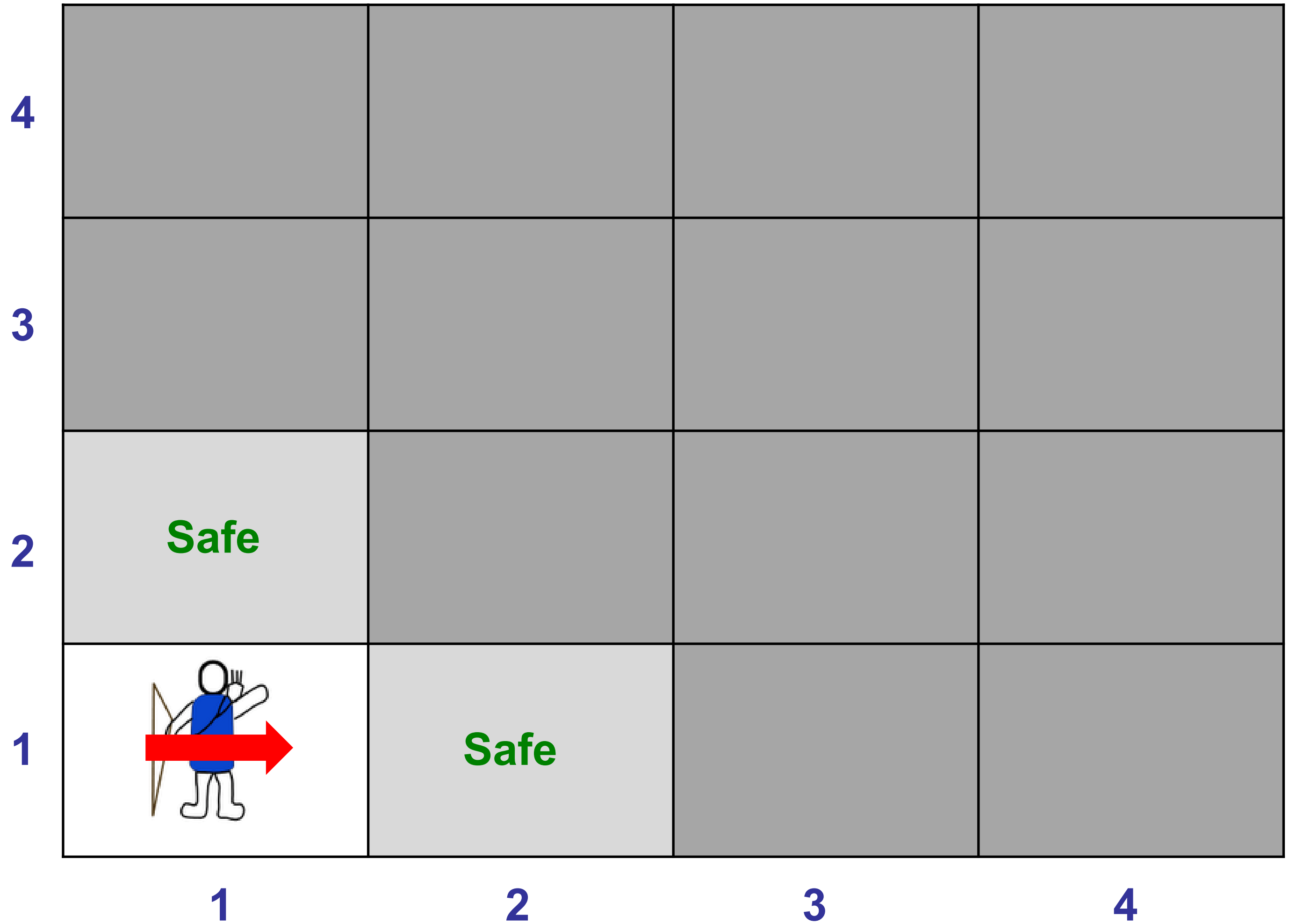
Example Wumpus World



Knowledge-based Player

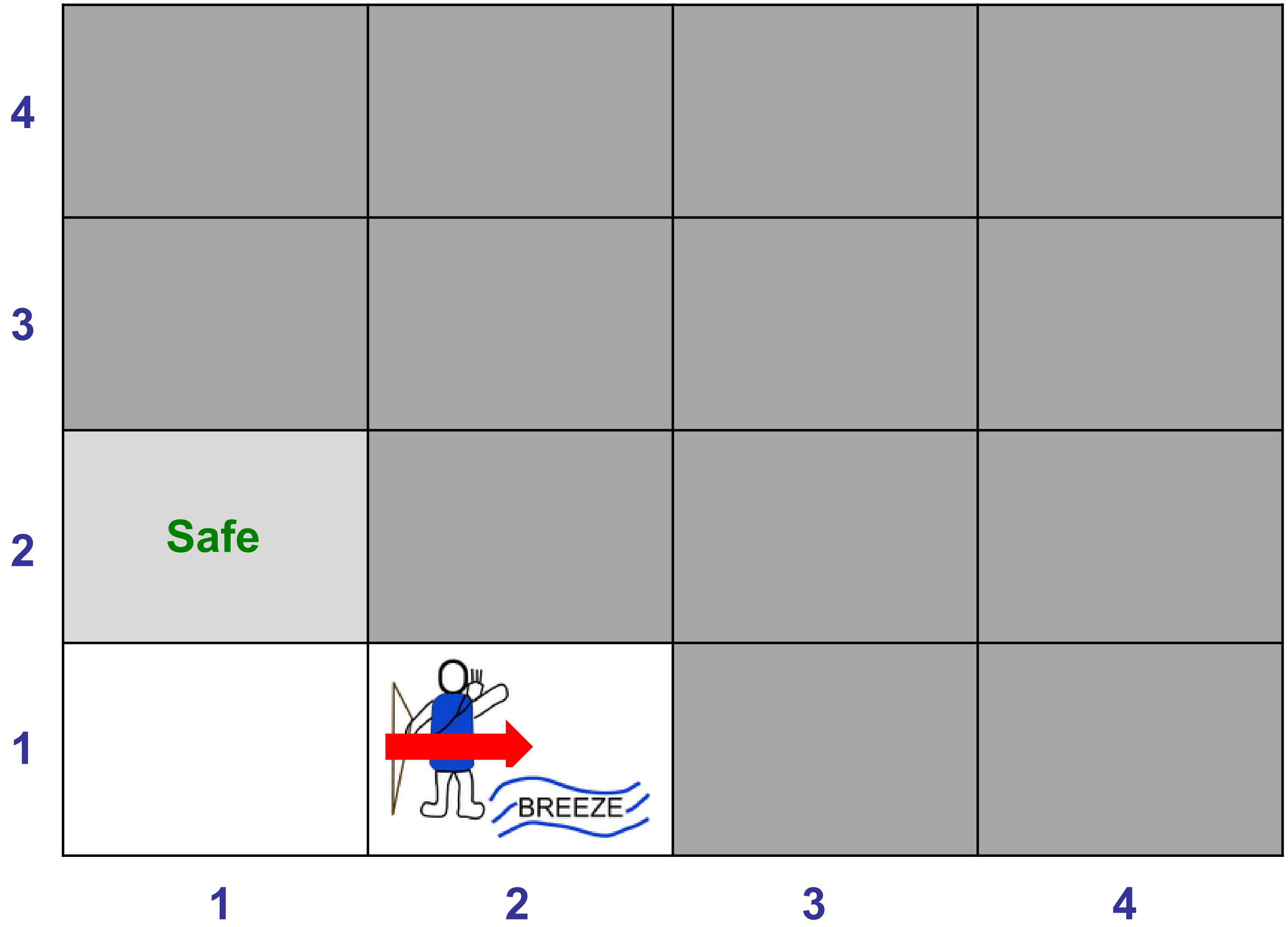
- Knowledge:
  - Player is in (1,1)
  - $\text{Percept}_{(1,1)}$ = [None, None, None, None, None]
- We can then conclude that the neighboring nodes (1,2) and (2,1) are safe!
- Now, lets start exploring the world

$\text{Percept}_{(1,1)}$ = [None, None, None, None, None]



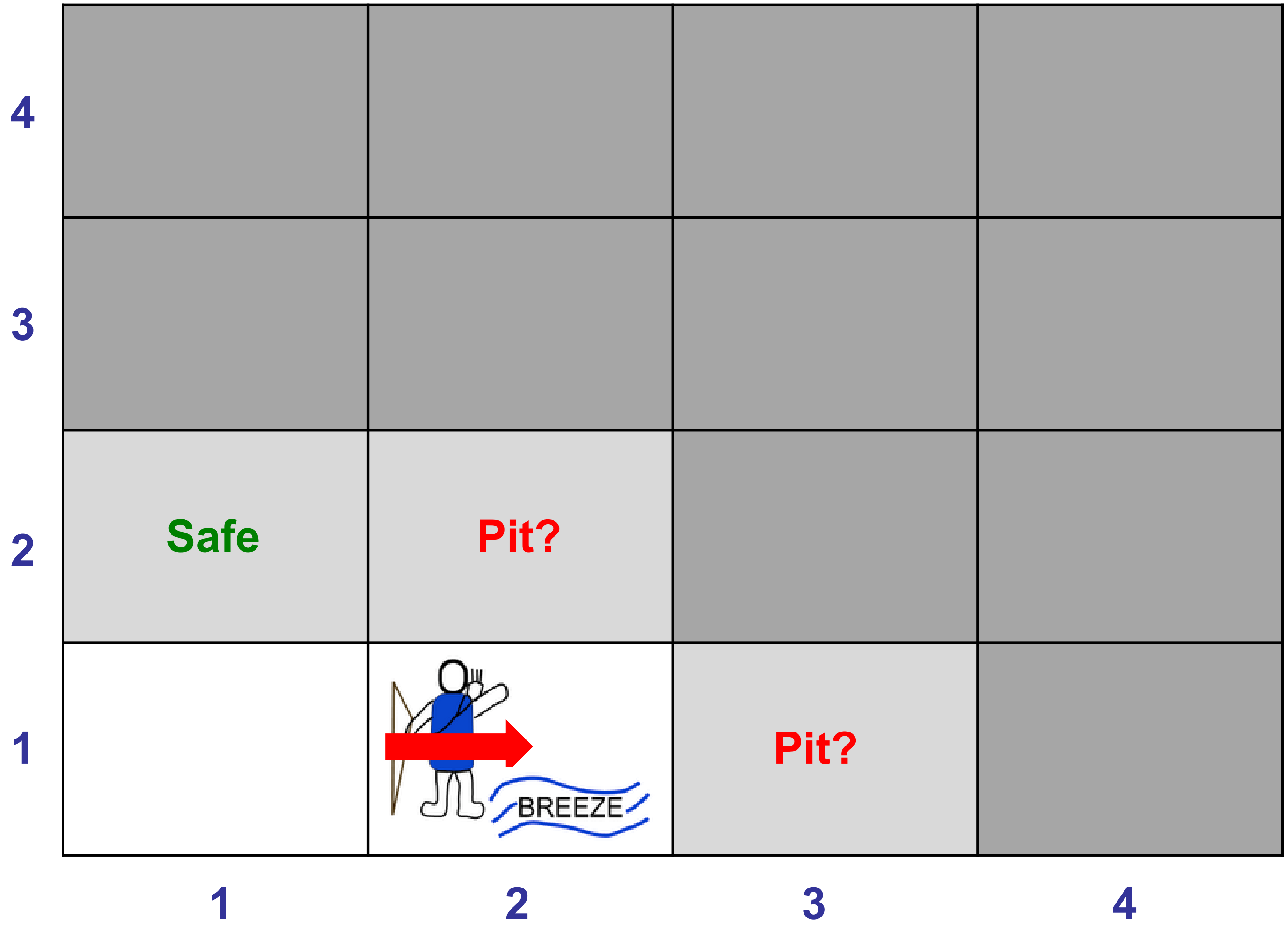
Action: Move Forward

$\text{Percept}_{(2,1)}$ = [None, Breeze, None, None, None]



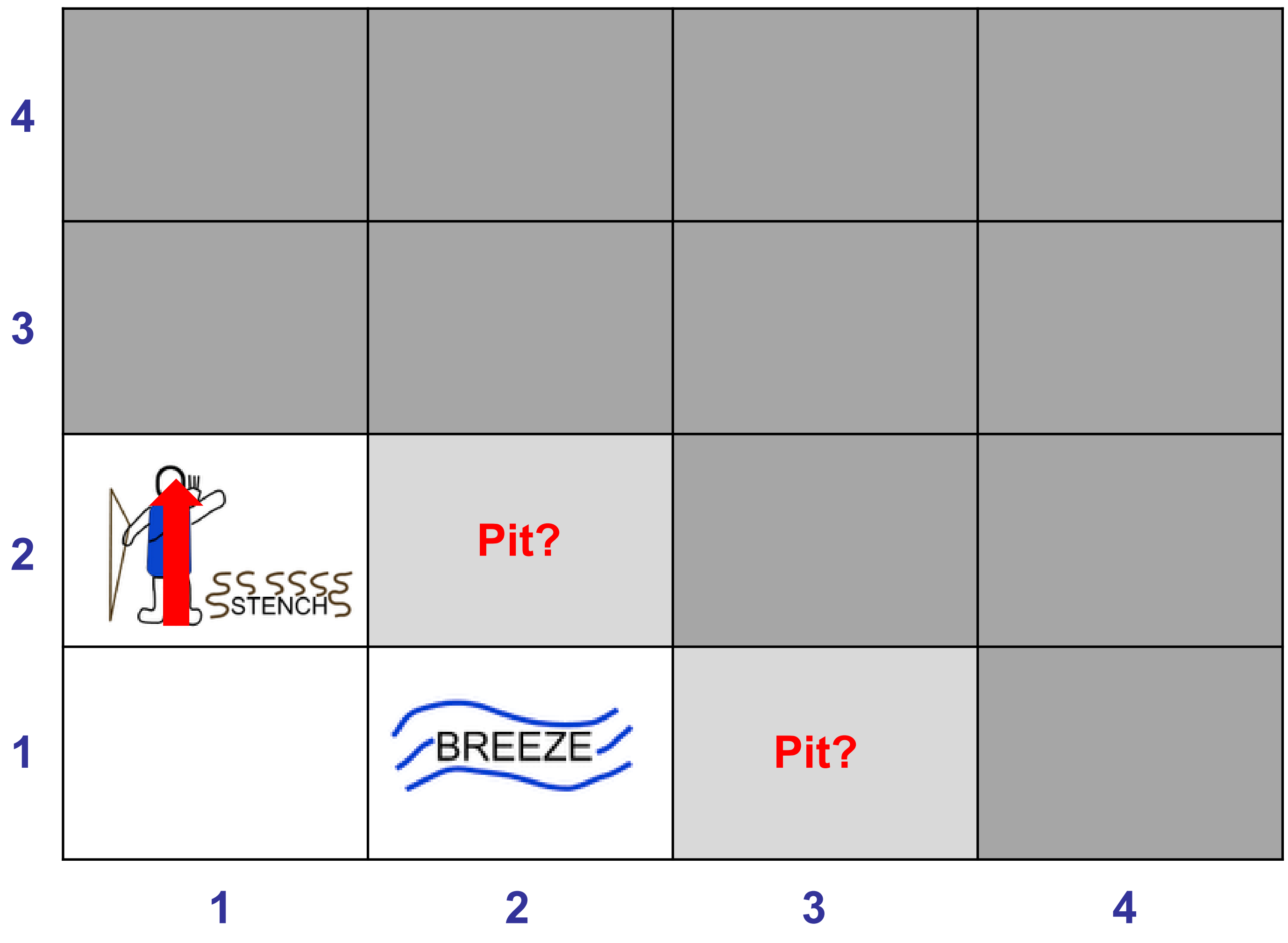
What conclusions can we make?

Percept$_{(2,1)}$ = [None, Breeze, None, None, None]



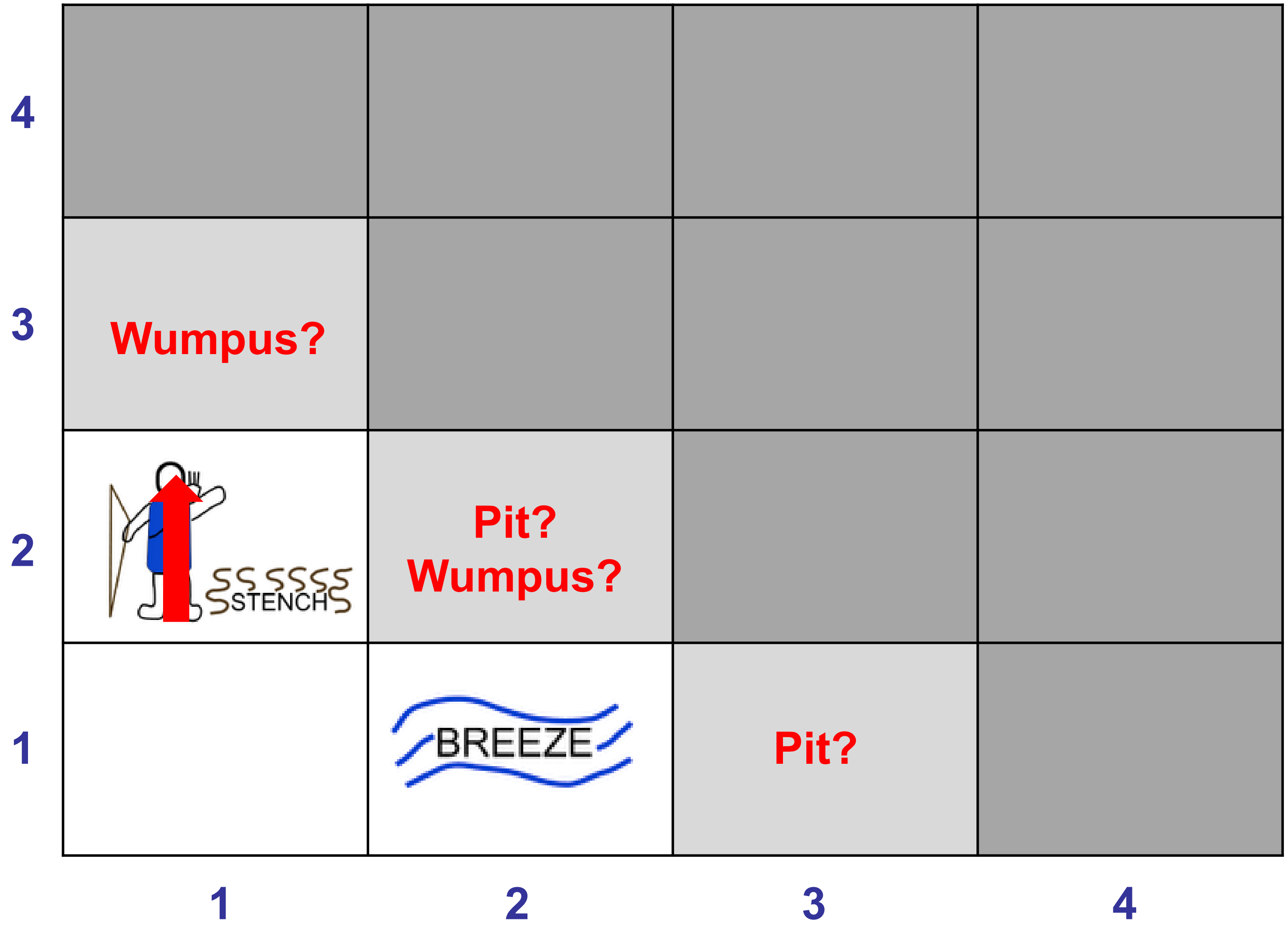
We need more information…

Percept$_{(1,2)}$ = [Stench, None, None, None, None]



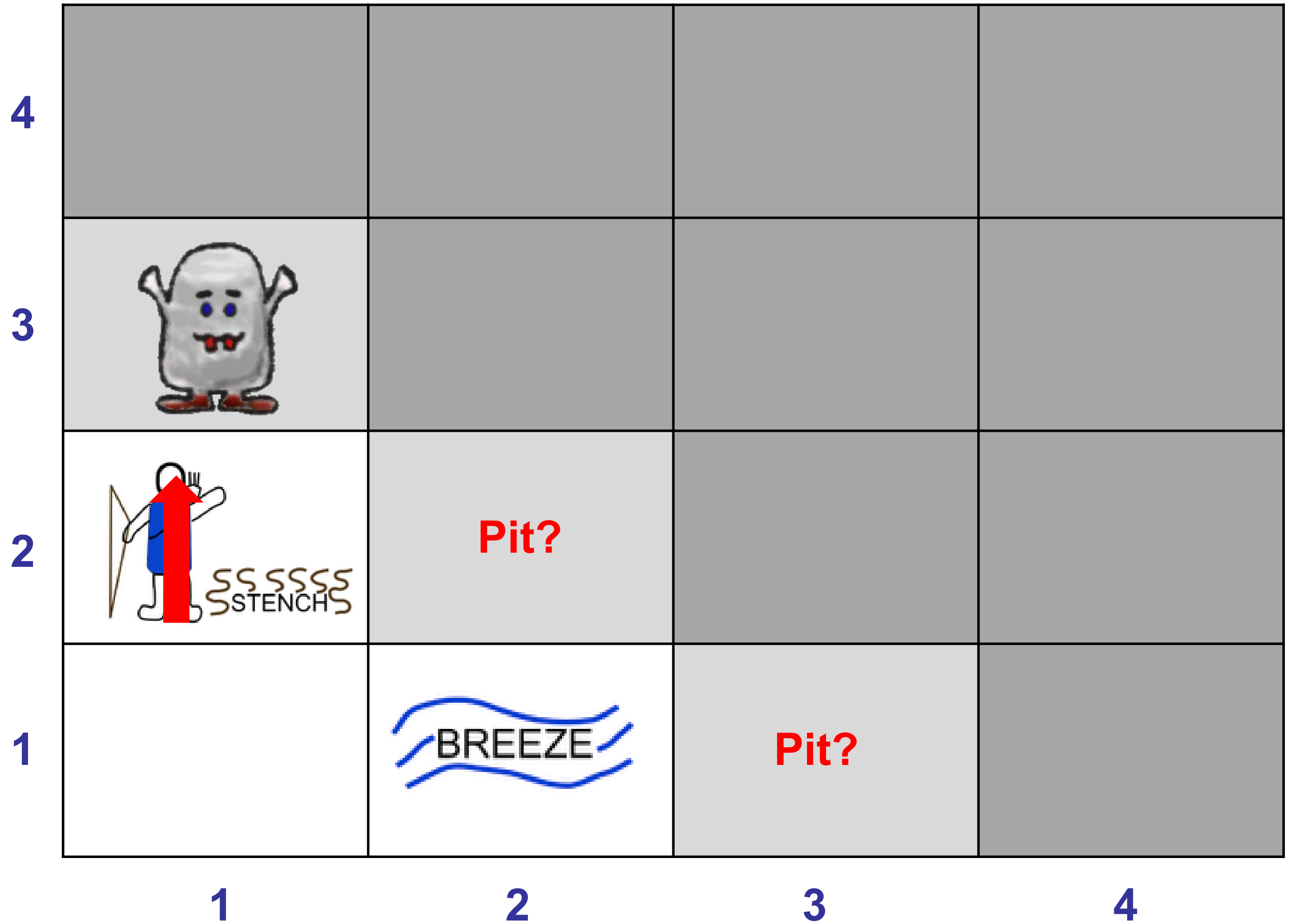
What conclusions can we make?

Percept$_{(1,2)}$ = [Stench, None, None, None, None]



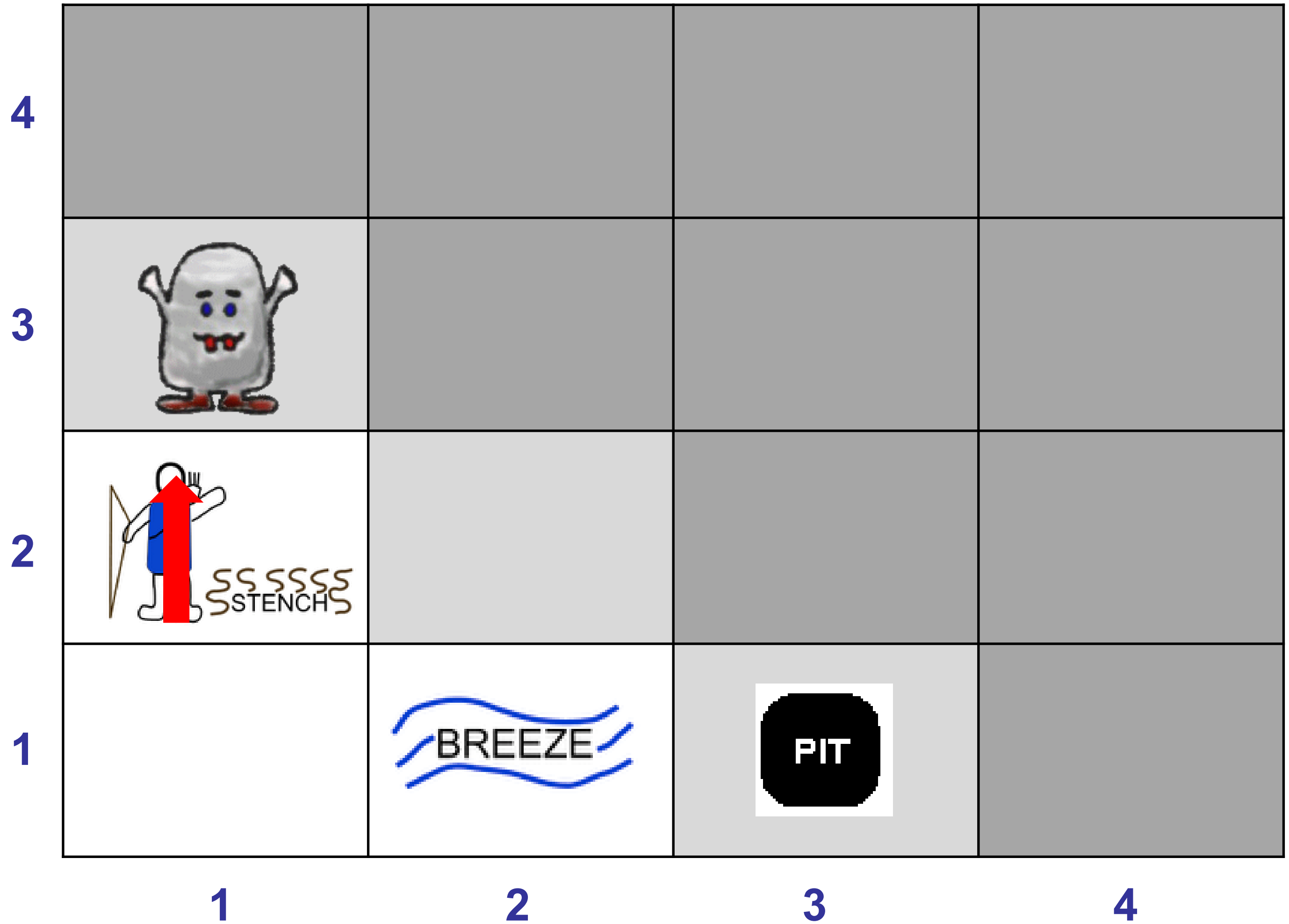
The Wumpus is nearby, but where?

$$\text{Percept}_{(1,2)} = [\text{Stench, None, None, None, None}]$$



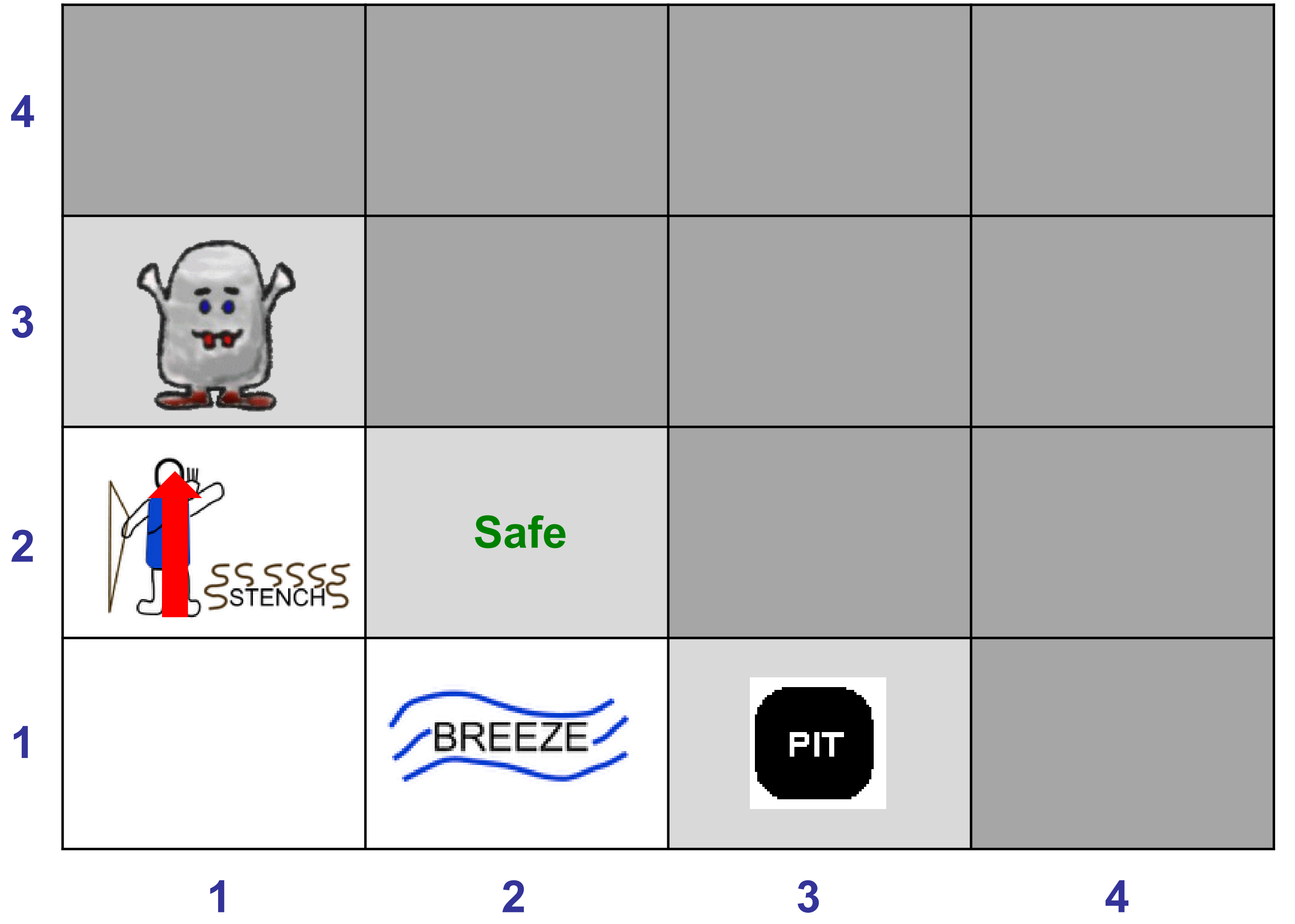
Wumpus must be in (1,3), since no stench was perceived in (2,1)

$\text{Percept}_{(1,2)}$ = [Stench, None, None, None, None]



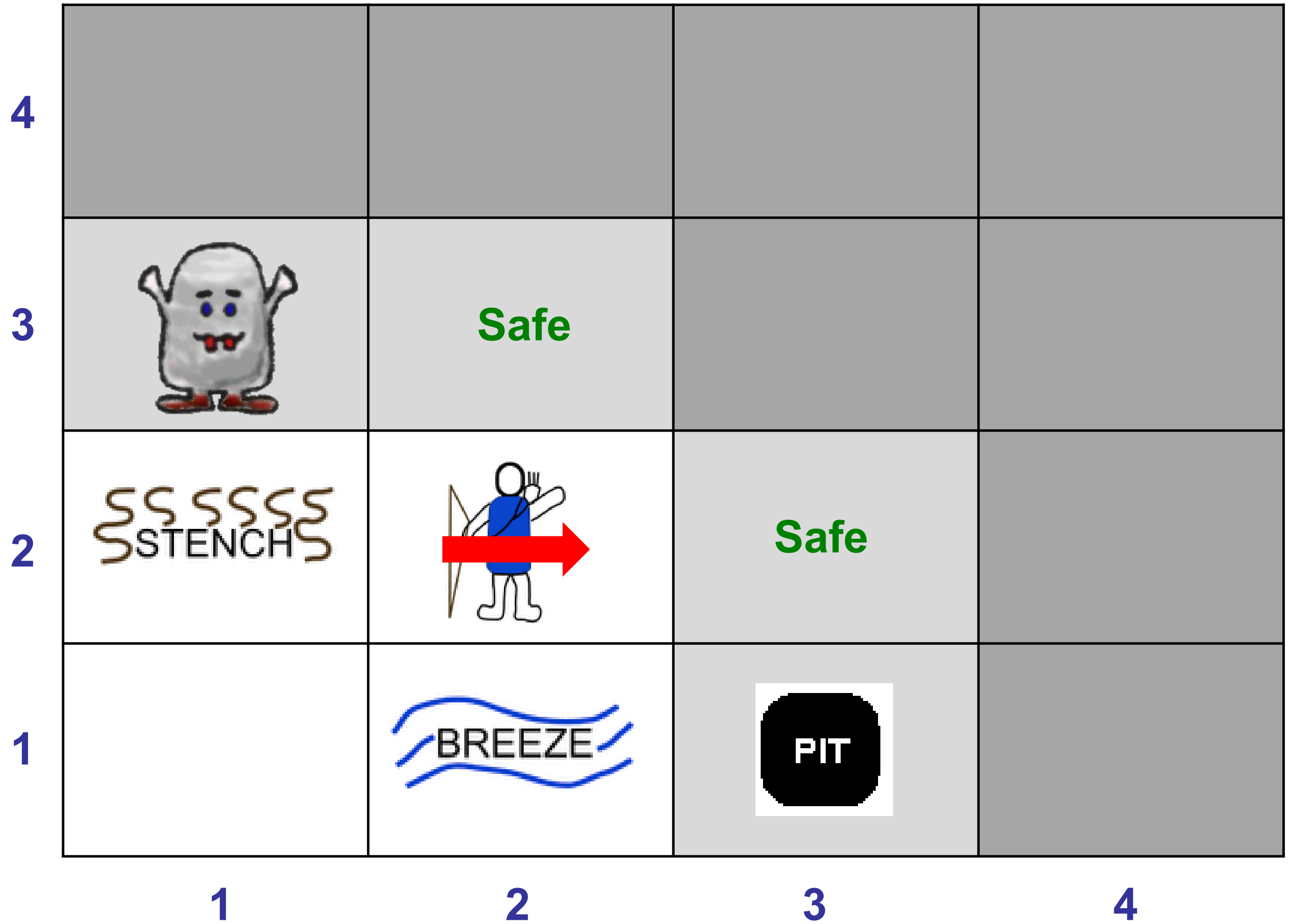
We can also conclude that the Pit must be in (3,1), since no Breeze is perceived in (1,2).

Percept$_{(1,2)}$ = [Stench, None, None, None, None]



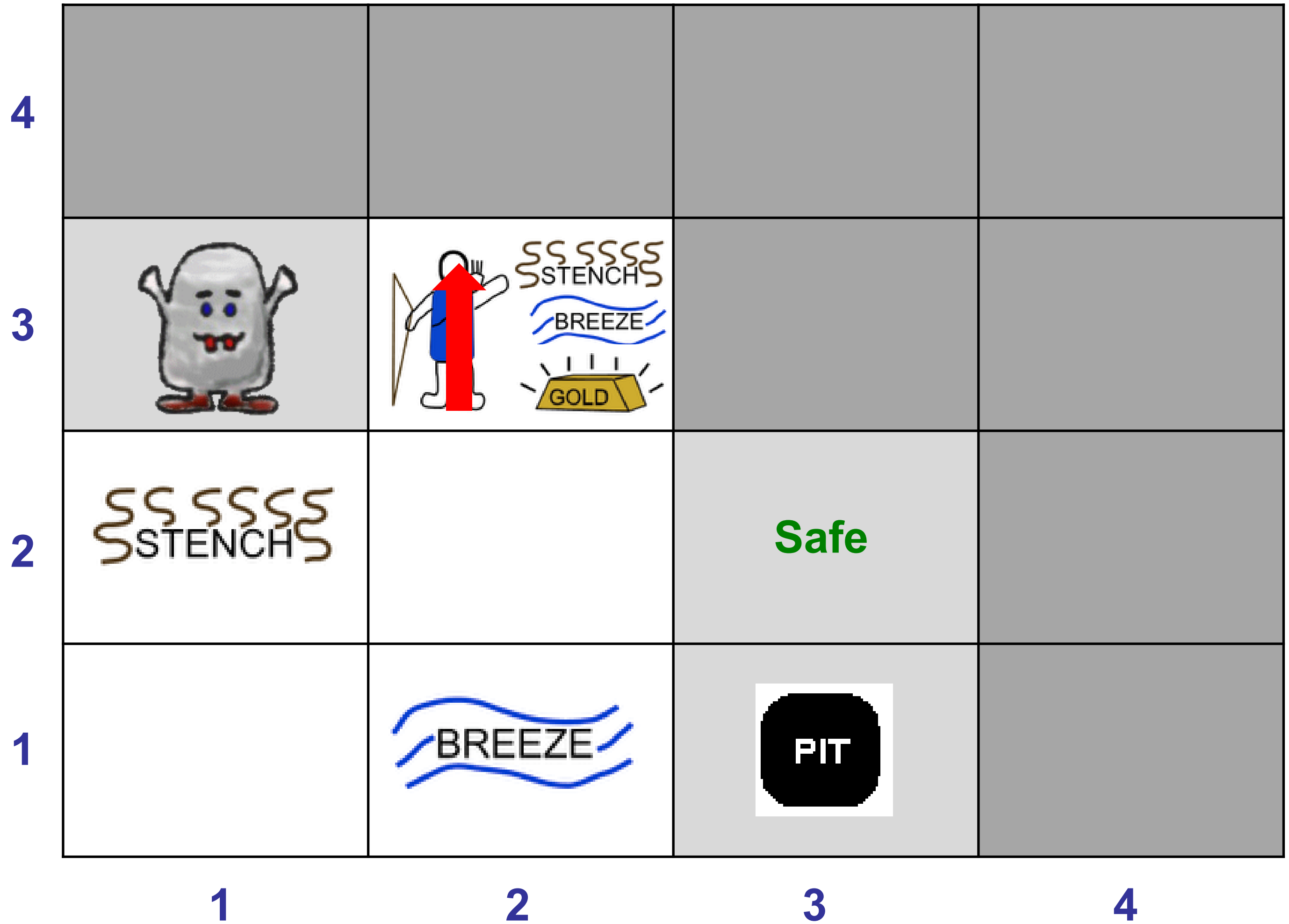
And that (2,2) is safe, since no Breeze is perceived and we know where Wumpus is.

Percept$_{(2,2)}$ = [None, None, None, None, None]



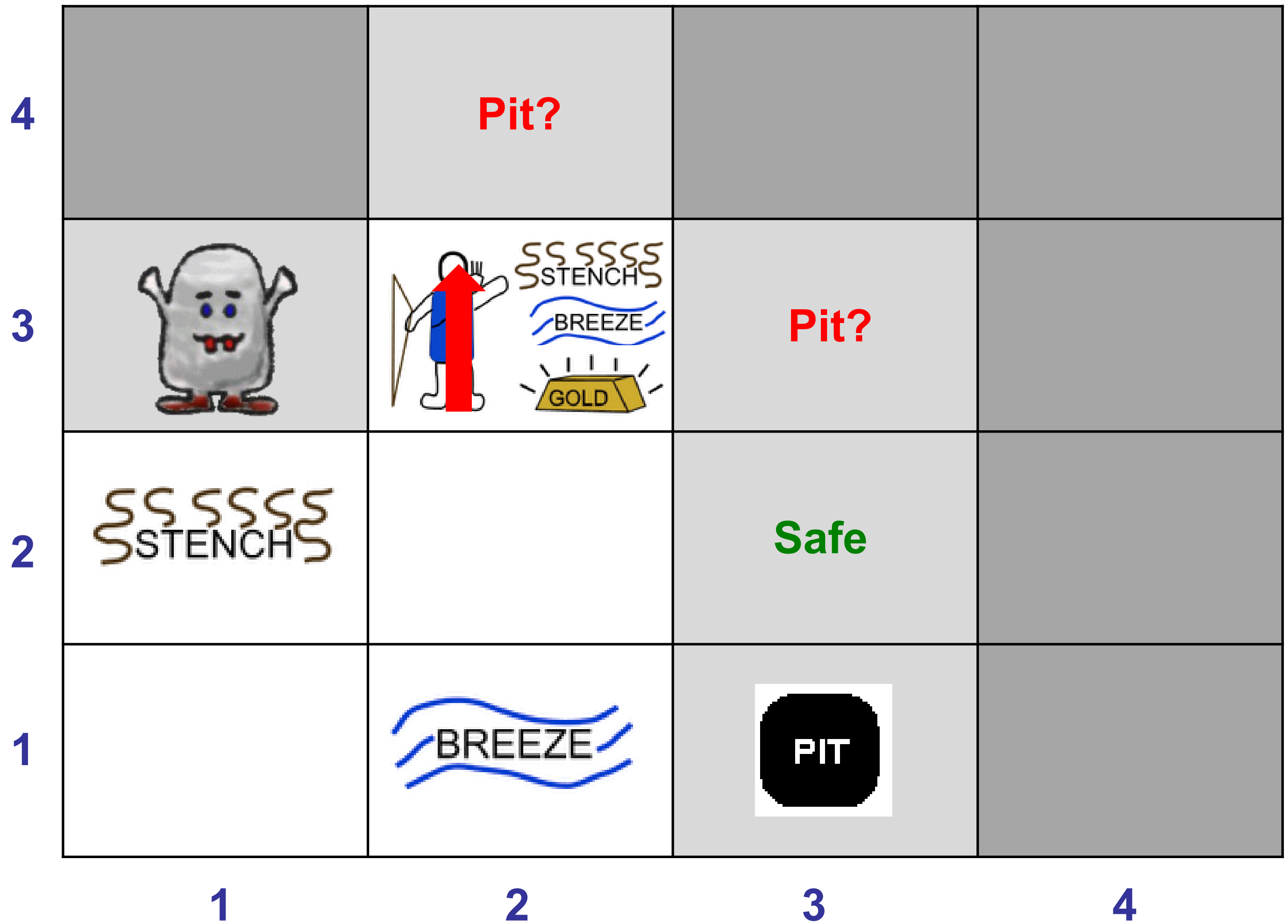
No Stench or Breeze, so (2,3) and (3,2) must be safe! We choose (2,3).

Percept$_{(2,3)}$ = [Stench, Breeze, Glitter, None, None]



We sense Glitter, so lets dig up the treasure!

Percept$_{(2,3)}$ = [Stench, Breeze, Glitter, None, None]



We can also draw the conclusion that there might be pits in (2,4) and (3,3).

Now, how can we formulate this in logic?

Logic in general

- Logic is a formal language for representing information so that conclusions can be drawn.
- A logic has two components:
    - Syntax: Defines which sentences that belong to the language
    - Semantics: Defines the meaning of the sentences (the truth of a sentence in a model)

Syntax and Semantics

- Example: The language of arithmetic
- Syntax:
  - $x + y > 2$ is a sentence
  - $y + y2 >$ is not a sentence
- Semantics:
  - $x + 2 > y$ is true if and only if the sum of x and 2 is larger than y.
  - $x + 2 > y$ is true in a model where $x = 3$ and $y = 1$
  - $x + 2 > y$ is false in a model where $x = 1$ and $y = 3$

Propositional Logic

- A very simple logic.
  - Also called Boolean Logic from George Boole
- Atomic Sentences, facts, are presented with a single Proposition Symbol.
  - A means ”box A is on the table”
  - $L_{(1,1)}$ means ”the player is in square (1,1).
- Complex sentences are constructed using the logical connectives:
  - $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$

Logical Connectives

- $\neg$ (not) negates a sentence.
  - $\neg$A means “box A is not on the table”
  - Literal: an atomic sentence (positive literal) or a negated atomic sentence (negative literal).
- $\wedge$ (and)
  - $A \wedge B$ “box A and box B is on the table”
  - A sentence with *and* is called a conjunction.
  - The parts are called conjuncts.

Logical Connectives

- $\vee$ (or)
  - $A \vee B$ “box A or box B is on the table”
  - A sentence with *or* is called a disjunction.
  - The parts are called disjuncts.
- $\Rightarrow$ (implies)
  - $A \Rightarrow \neg B$ “if box A is on the table, box B is not”
  - A sentence with *implies* is called an implication.
  - Left part: premise or antecedent.
  - Right part: Conclusion or consequent.
  - Rules or if-then statements.
  - $\Rightarrow$ is sometimes written $\rightarrow$ or $\supset$

Logical Connectives

- $\Leftrightarrow$ (if and only if, biconditional)
  - $A \Leftrightarrow \neg B$ "box A is on the table if and only if box B is not"
  - $A \Rightarrow \neg B$ is a weaker statement. In this case box A can be on the table under other circumstances than if B is not.
  - If stated as $A \Leftrightarrow \neg B$, box A is on the table only if B is not. No other circumstances are valid.

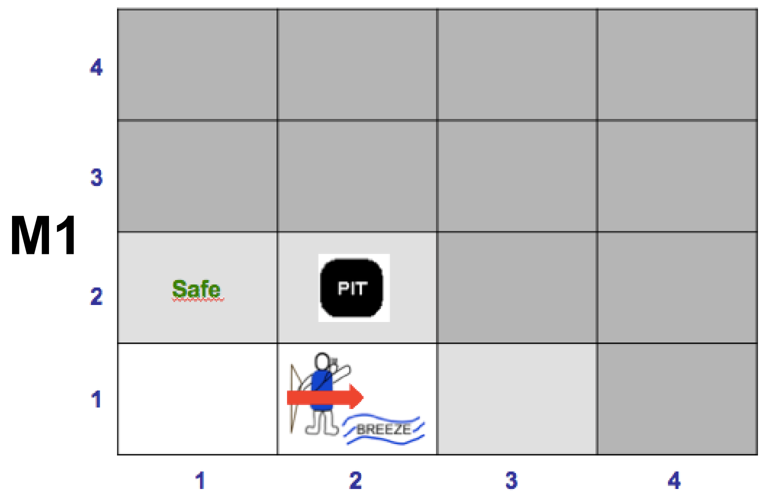
Syntax

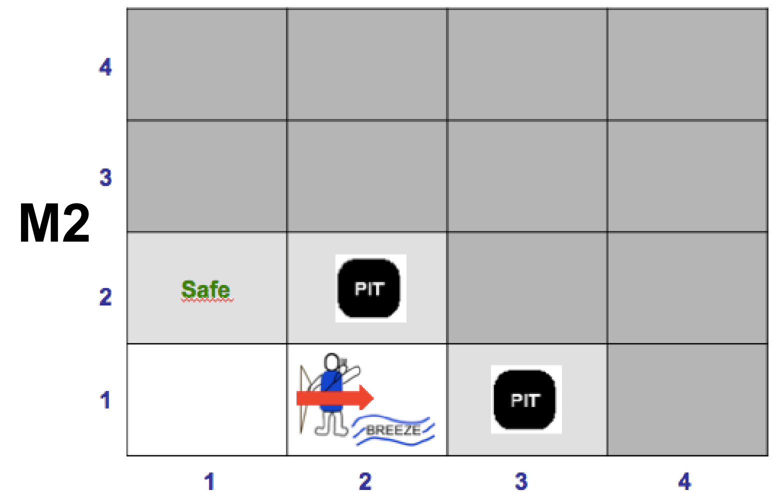
- Every sentence with binary connectives must be enclosed in parentheses:
  - $A \wedge B \Rightarrow C$ (Wrong)
  - $((A \wedge B) \Rightarrow C)$ Correct
- This makes sentences a bit difficult to read, so parentheses are often omitted.
- Instead relying on order of precedence:
  - In arithmetic * and / have a higher order than + or – and is calculated first.
  - Order in logic: $\neg$, $\wedge$, $\vee$, $\Rightarrow$ and $\Leftrightarrow$

Semantics

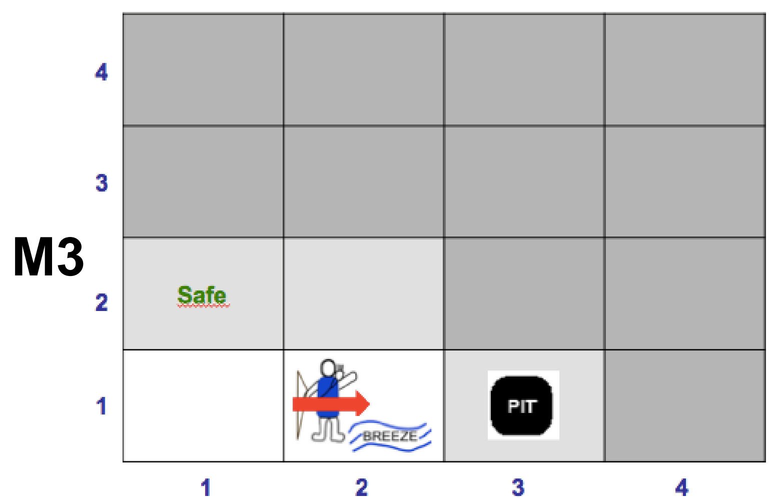
- Defines the rules for determining the *truth of a sentence* with respect to a *particular model*.
- Can be true, or false.
- Example:
    - Proposition symbols: A, B, C
    - [false, true, false], [true, true, false], …
    - $2^3 = 8$ possible combinations (= models).
- Note that a proposition is just a symbol, it can mean anything.

Semantics





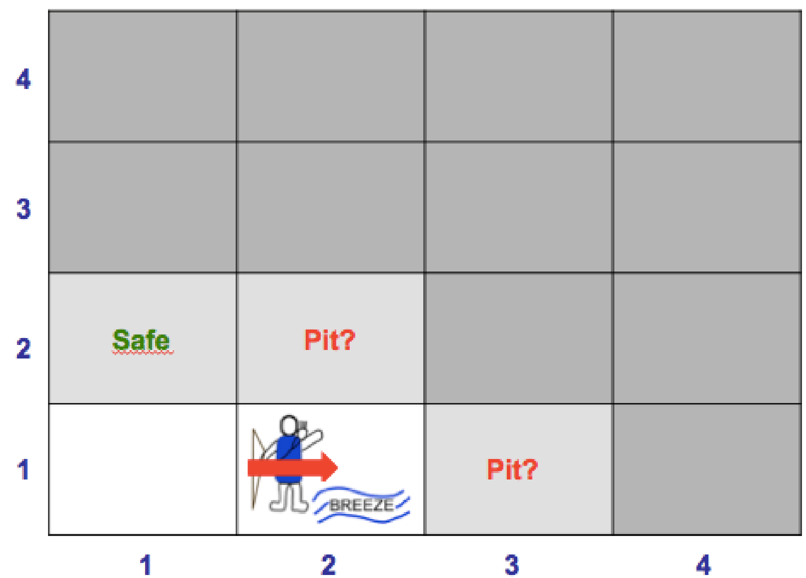
- Three possible models!
- $P_{(2,2)}$ is true in two models.



Semantics

- Atomic sentences:
  - If true, the proposition must be true in all models.
  - If false, the proposition must be false in all models.

- $P_{(2,2)}$ is not valid.
- $P_{(3,1)}$ is not valid.
- $B_{(2,1)}$ is valid.



Truth table

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
F	F					
F	T					
T	F					
T	T					

Truth table

P	Q	$\neg$P	P $\wedge$ Q	P $\vee$ Q	P $\Rightarrow$ Q	P $\Leftrightarrow$ Q
F	F	T				
F	T	T				
T	F	F				
T	T	F				

Just negate P.

Truth table

P	Q	$\neg$P	P $\wedge$ Q	P $\vee$ Q	P $\Rightarrow$ Q	P $\Leftrightarrow$ Q
F	F	T	F			
F	T	T	F			
T	F	F	F			
T	T	F	T			

True if both P and Q is true, otherwise false.

Truth table

P	Q	$\neg$P	P $\wedge$ Q	P $\vee$ Q	P $\Rightarrow$ Q	P $\Leftrightarrow$ Q
F	F	T	F	F		
F	T	T	F	T		
T	F	F	F	T		
T	T	F	T	T		

True if <u>either P or Q</u> is true, otherwise false.

Truth table

P	Q	$\neg$P	P $\wedge$ Q	P $\vee$ Q	P $\Rightarrow$ Q	P $\Leftrightarrow$ Q
F	F	T	F	F	T	
F	T	T	F	T	T	
T	F	F	F	T	F	
T	T	F	T	T	T	

This one is a bit tricky. Think of it like this:

"*If P is true, then I am claiming that Q is true. Otherwise I make no claim*"

Always true if the antecedent is false (I make no claim).

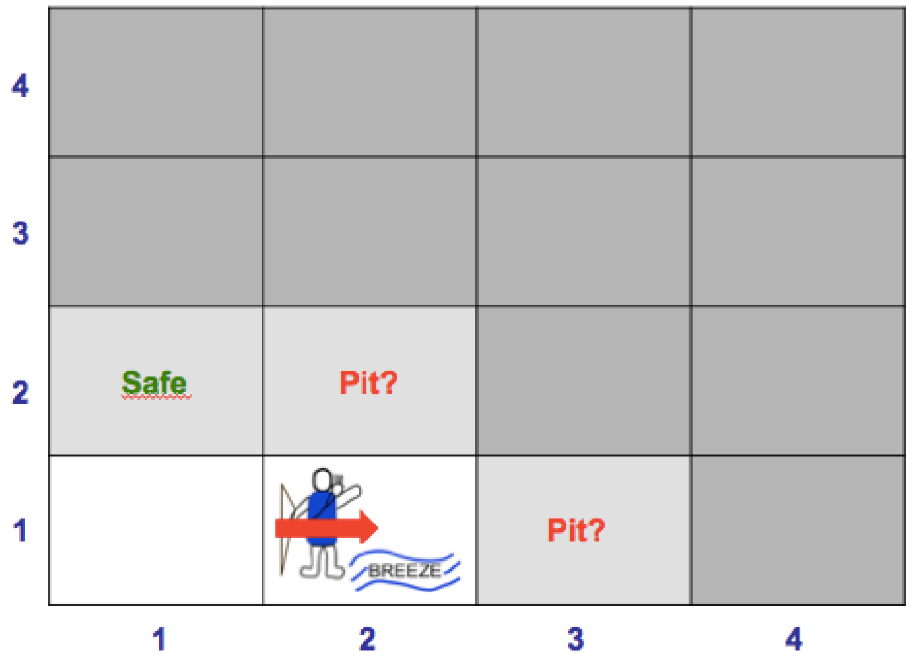
Truth table

P	Q	$\neg$P	P $\wedge$ Q	P $\vee$ Q	P $\Rightarrow$ Q	P $\Leftrightarrow$ Q
F	F	T	F	F	T	T
F	T	T	F	T	T	F
T	F	F	F	T	F	F
T	T	F	T	T	T	T

True if both P $\Rightarrow$ Q and Q $\Rightarrow$ P is true. Often written as ”P if and only if Q” or sometimes ”P iff Q”.

Basically, true if left and right parts are equal.

Back to the Wumpus World

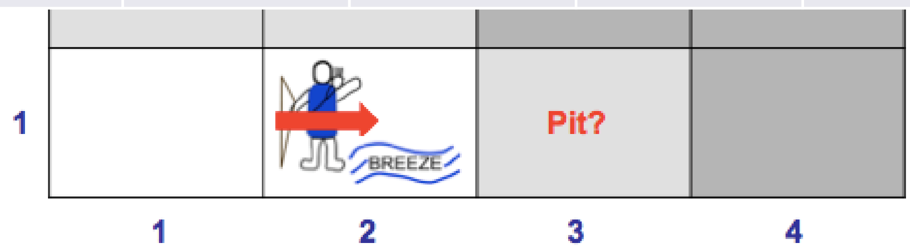


Can be written as:

$B_{(2,1)} \Leftrightarrow P_{(2,2)} \vee P_{(3,1)}$

Back to the Wumpus World

P	Q	¬P	P ∧ Q	P ∨ Q	P ⇒ Q	P ⇔ Q
F	F	T	F	F	$\boxed{T}$	T
F	T	T	F	T	$\boxed{T}$	F
T	F	F	F	T	F	F
T	T	F	T	T	T	T



Note that:

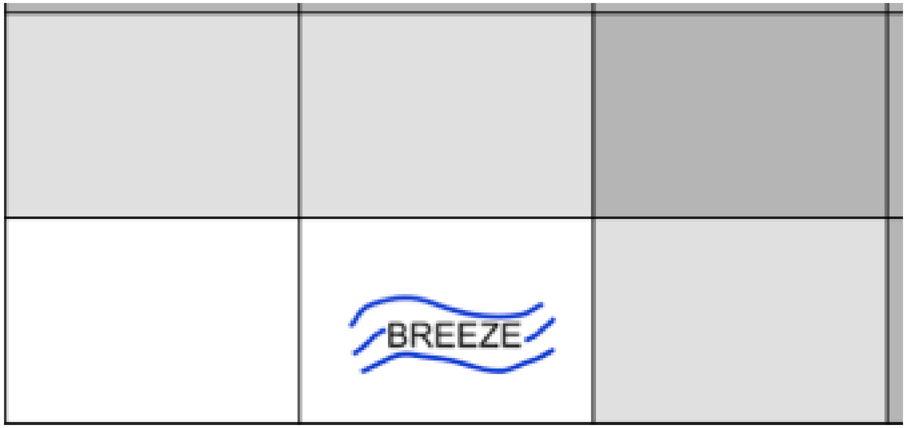
$B_{(2,1)} \Rightarrow P_{(2,2)} \vee P_{(3,1)}$

is **not entirely correct**, since according to the truth table $B_{(2,1)}$ can be false while $P_{(2,2)} \vee P_{(3,1)}$ is true, which violates the game rules.

Wumpus Knowledge Base

Facts:

- $B_{(2,1)}$
- $\neg B_{(1,2)}$
- $\neg P_{(1,1)}$
- $\neg P_{(2,1)}$



Rules:

- $B_{(1,1)} \Leftrightarrow P_{(1,2)} \lor P_{(2,1)}$
- $B_{(2,1)} \Leftrightarrow P_{(1,1)} \lor P_{(2,2)} \lor P_{(3,1)}$

Wumpus Knowledge Base

Two possible conclusions to check for safe moves:

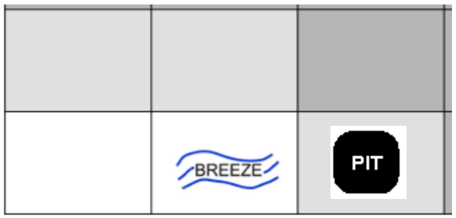
- α_1 = There is no pit in (1,2)
- α_2 = There is no pit in (2,2)

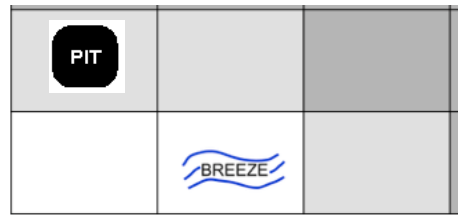
How can we check if α_1 and α_2 is true?

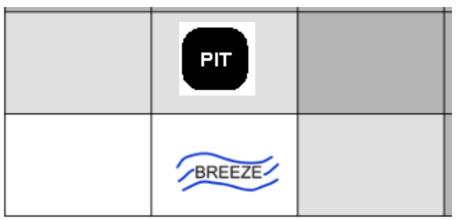
Entailment

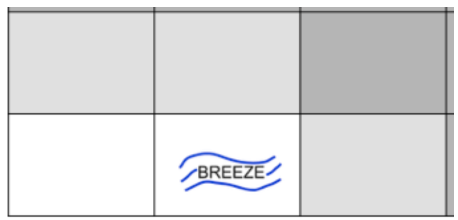
- Definition:
  - $\alpha \vDash \beta$
  - Means that ”α entails β if and only if, in every model in which α is true, then β must also be true”
- We can use entailment to derive conclusions (=logical inference):
  - $KB \vDash \alpha_1$
  - $KB \vDash \alpha_2$
- Define all possible models, and check in which models KB entails each conclusion.
  - Model checking

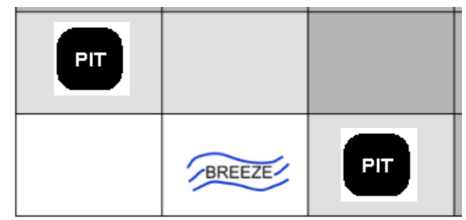
Possible models.
Which models match KB?

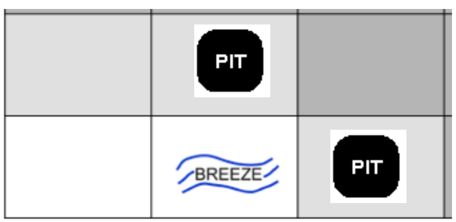


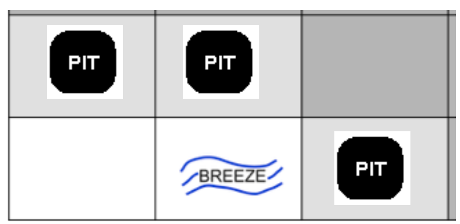


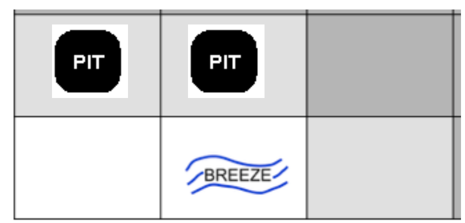


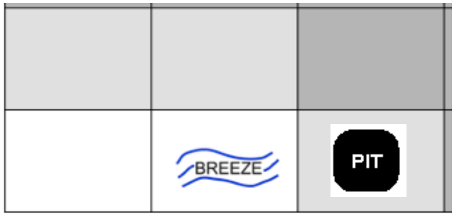












<u>KB</u>

Facts:

1. $B_{(2,1)}$
2. $\neg B_{(1,2)}$
3. $\neg P_{(1,1)}$
4. $\neg P_{(2,1)}$

Rules:

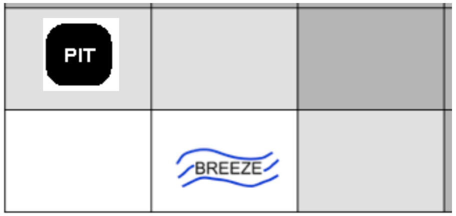
1. $B_{(1,1)} \Leftrightarrow P_{(1,2)} \vee P_{(2,1)}$
2. $B_{(2,1)} \Leftrightarrow P_{(1,1)} \vee P_{(2,2)} \vee P_{(3,1)}$

New facts:

- $P_{(3,1)}$

- Does not contradict any fact in KB.
- Rule 1 does not include $P_{(3,1)}$
- Matches Rule 2:
  - $B_{(2,1)}$ is true, and $P_{(3,1)}$ is true

The model matches KB!



<u>KB</u>

Facts:

1. $B_{(2,1)}$
2. $\neg B_{(1,2)}$
3. $\neg P_{(1,1)}$
4. $\neg P_{(2,1)}$

Rules:

1. $B_{(1,1)} \Leftrightarrow P_{(1,2)} \vee P_{(2,1)}$
2. $B_{(2,1)} \Leftrightarrow P_{(1,1)} \vee P_{(2,2)} \vee P_{(3,1)}$

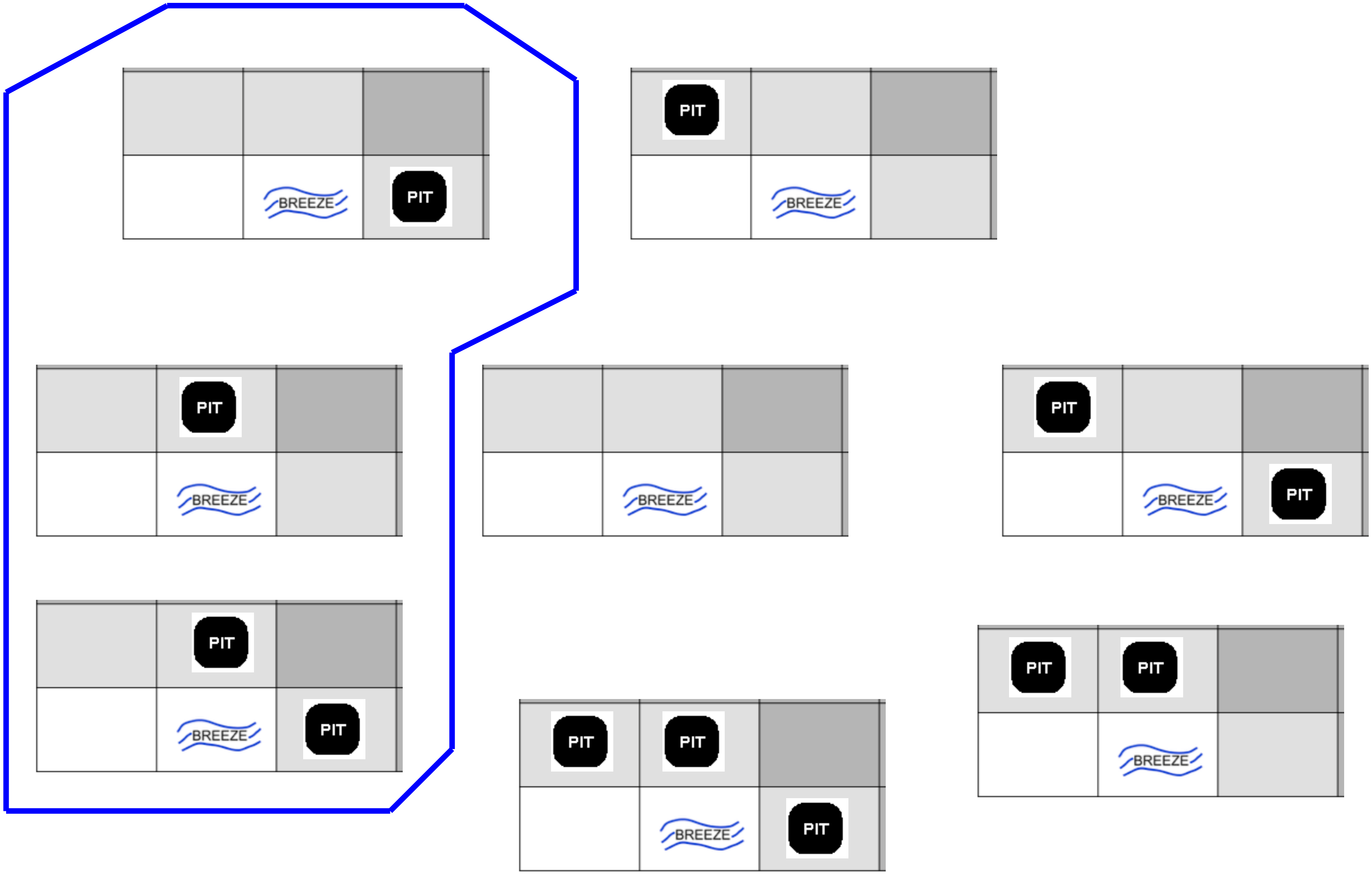
New facts:

- $P_{(1,2)}$

- Does not contradict to any fact in KB.
- Rule 2 does not include $P_{(1,2)}$
- Does <u>not</u> match Rule 1:
  - $B_{(1,1)}$ is false and $P_{(1,2)}$ is true

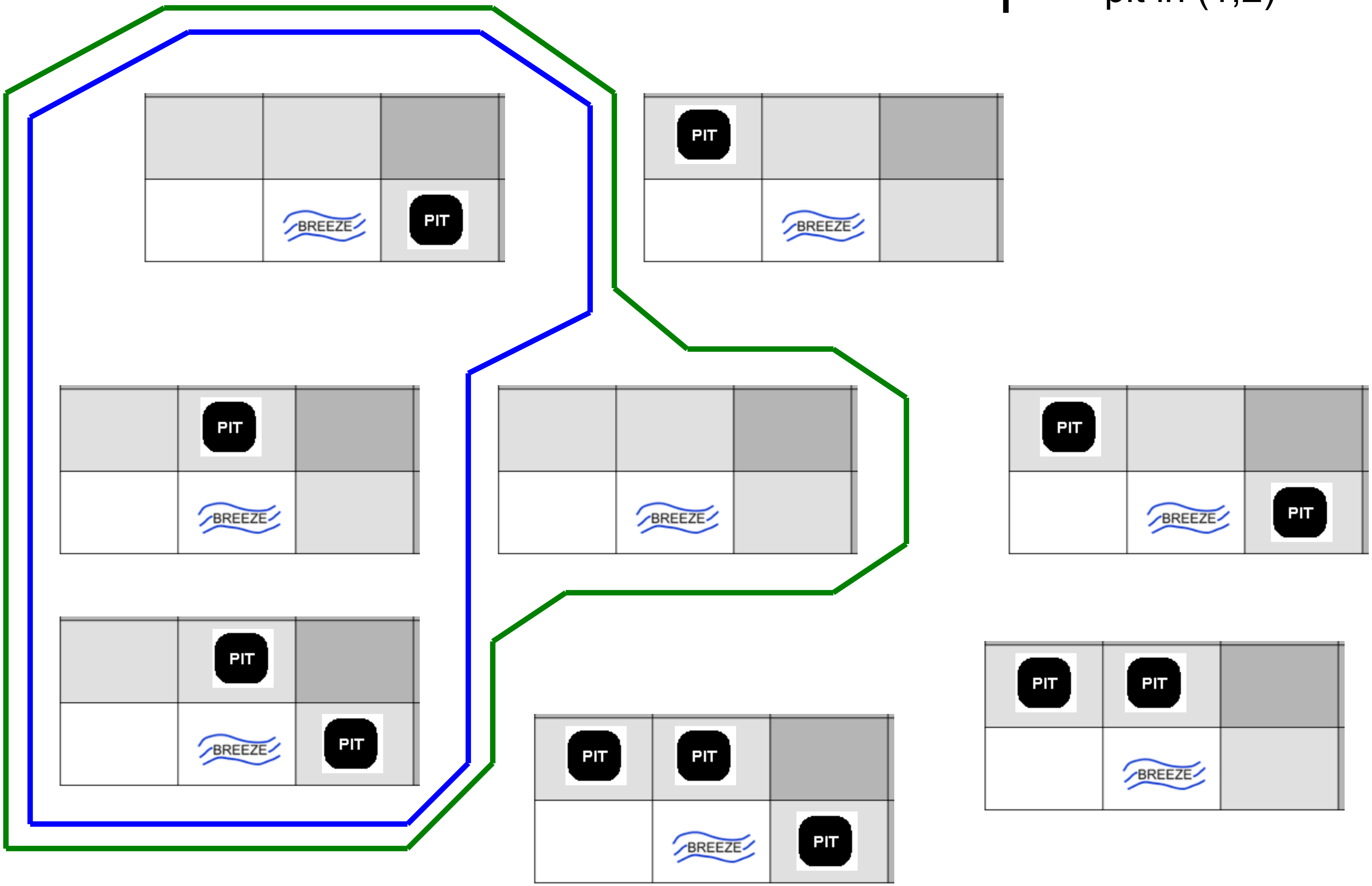
The model does not match KB!

Models in KB



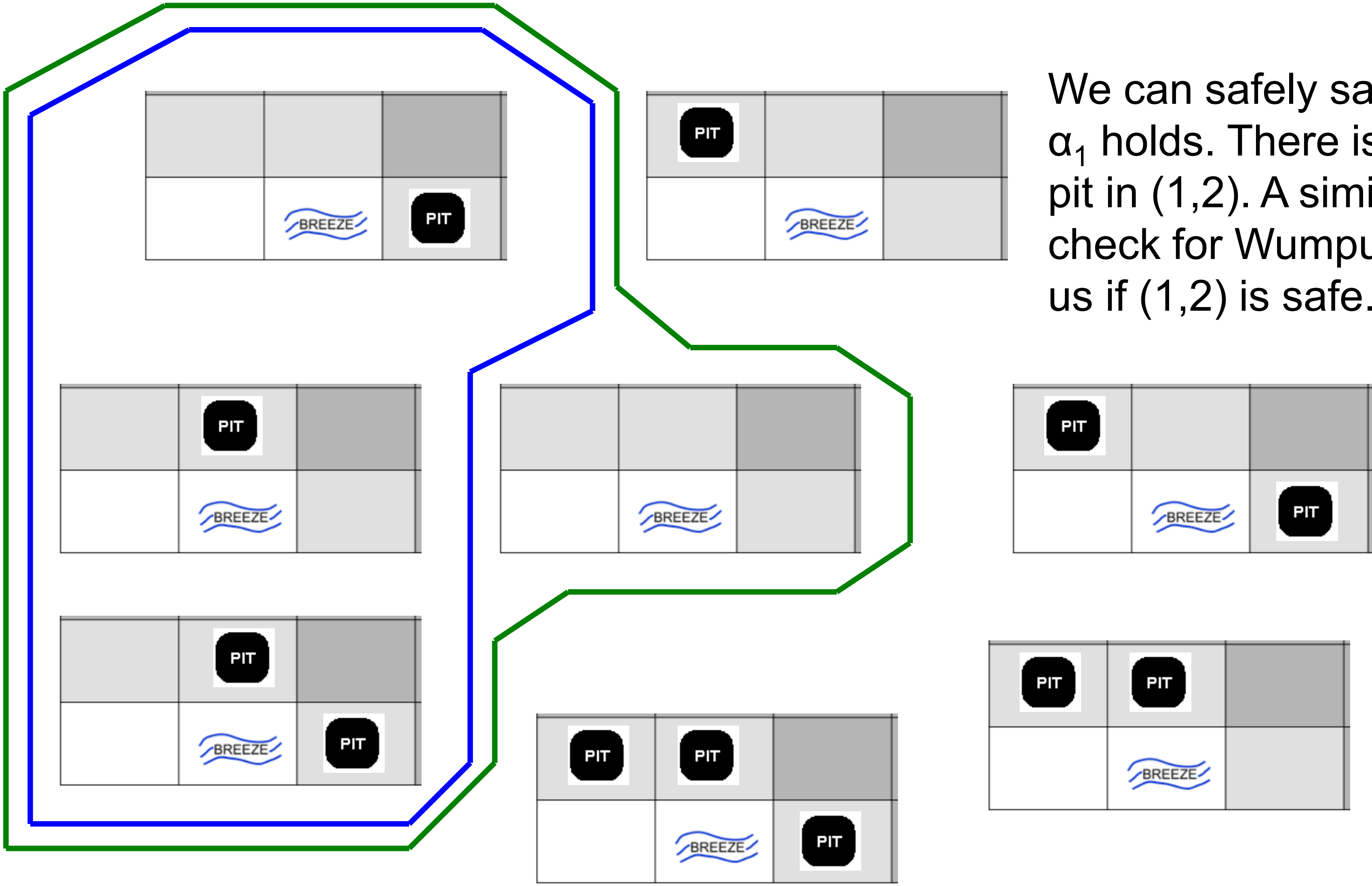
Models in α_1

α_1 = There is no pit in (1,2)



Definition:

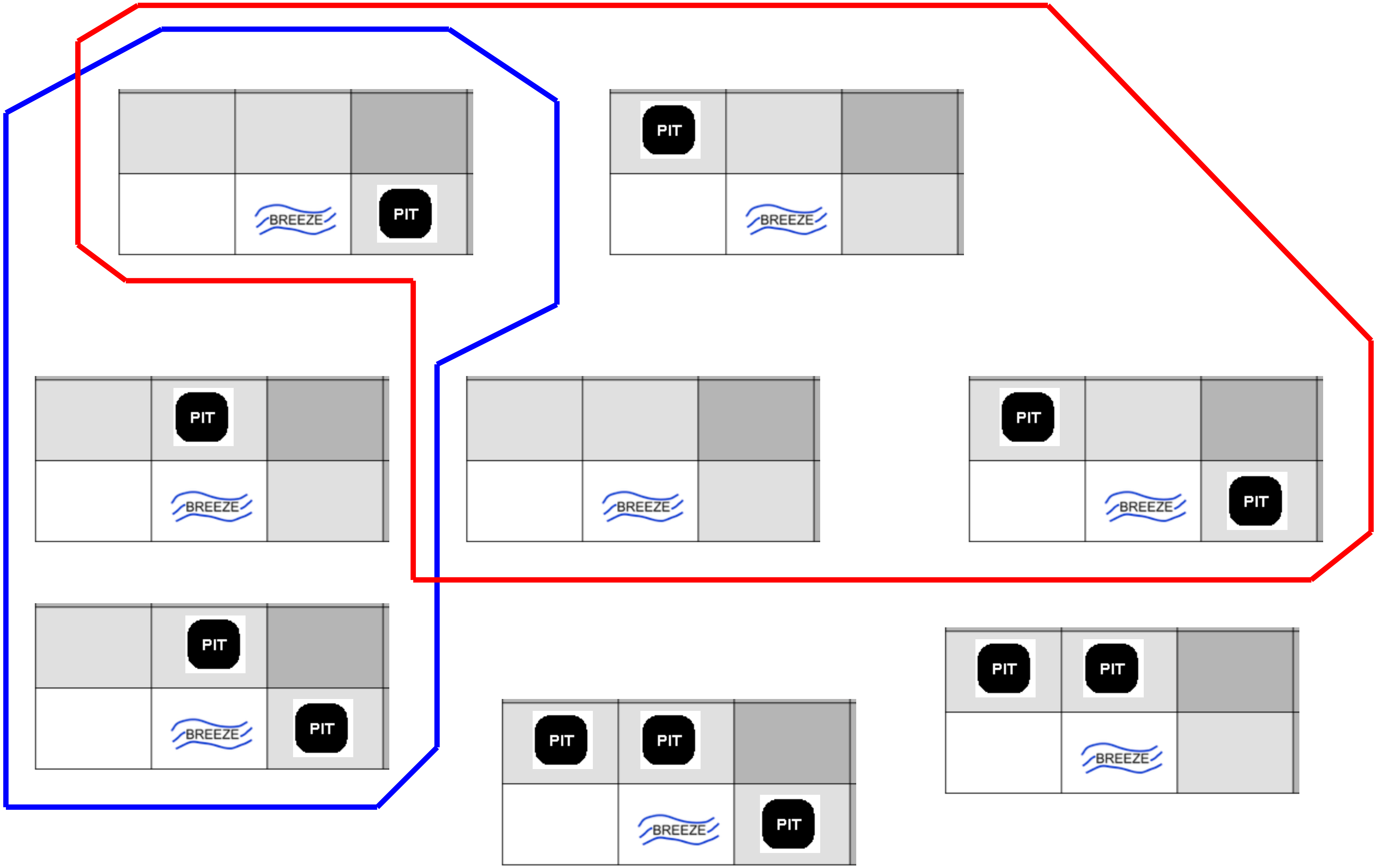
in every model in which KB is true, α_1 is also true



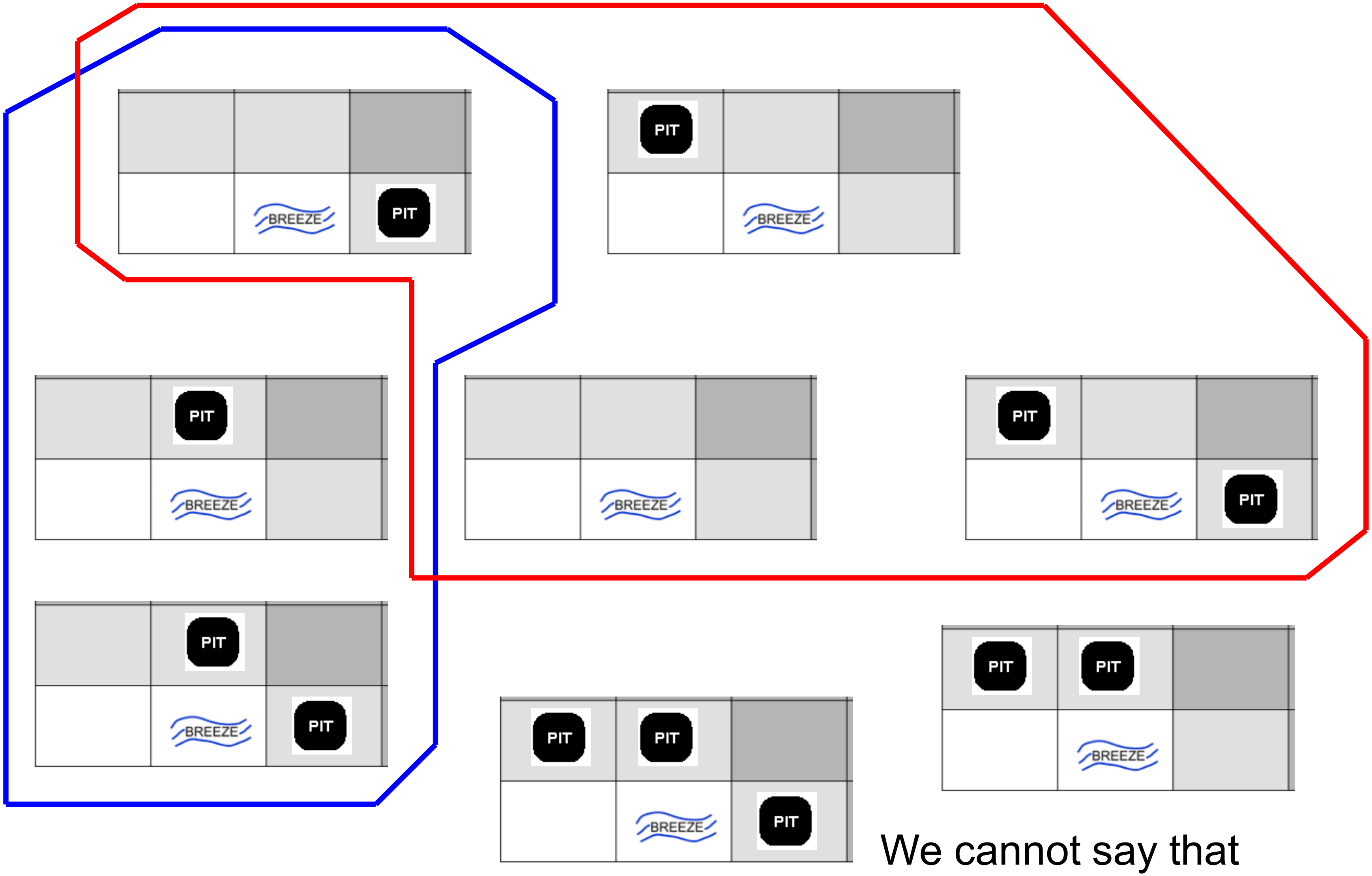
We can safely say that α_1 holds. There is no pit in (1,2). A similar check for Wumpus tells us if (1,2) is safe.

Models in α_2

α_2 = There is no pit in (2,2)



Two models in KB are not covered by α_2



We cannot say that there is no pit in (2,2).

Inference

- If an inference algorithm *i* can derive α from KB, we write:
  - KB $\vdash_i$ α
  - Means ”α is derived from KB by *i*”
- In our case:
  - ”α is derived from KB by *model checking*”

Inference

- Sound / truth-preserving:
  - An inference algorithm that derives only entailed sentences. No incorrect sentence can be derived.
- Completeness:
  - An inference algorithm is complete if it can derive any sentence that is entailed.
- Model checking is sound and complete, but is not feasible for very large knowledge bases.
    - Time complexity $O(2^n)$ if KB and α contain n symbols.

Inference

- The only real problem with model checking is performance. Exponential time complexity is bad…
- It does not work for infinite KBs.
- There are other, much more efficient algorithms.

- But…
- ”every known inference algorithm for propositional logic has a worst-case complexity that is exponential in the size of the input”

- Let’s take a look at some other inference algorithms.
- But first we need to learn some new concepts:
  - Equivalence
  - Validity
  - Satisfiability

Equivalence

- Two sentences are logically equivalent if they are true in the same set of models.
- We write this as:
  - $\alpha \equiv \beta$
  - … which is defined as:
  - *$\alpha \equiv \beta$ if and only if $\alpha \models \beta$ and $\beta \models \alpha$*
- There are lots of other useful equivalences.

Standard logical equivalences

$(\alpha \land \beta) \equiv (\beta \land \alpha)$	Commutativity of $\land$
$(\alpha \lor \beta) \equiv (\beta \lor \alpha)$	Commutativity of $\lor$
$((\alpha \land \beta) \land \gamma) \equiv (\alpha \land (\beta \land \gamma))$	Associativity of $\land$
$((\alpha \lor \beta) \lor \gamma) \equiv (\alpha \lor (\beta \lor \gamma))$	Associativity of $\lor$
$\neg(\neg\alpha) \equiv \alpha$	Double-negation elimination
$(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha)$	Contraposition
$(\alpha \Rightarrow \beta) \equiv (\neg\alpha \lor \beta)$	Implication elimination
$(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha))$	Biconditional elimination
$\neg(\alpha \land \beta) \equiv (\neg\alpha \lor \neg\beta)$	De Morgan
$\neg(\alpha \lor \beta) \equiv (\neg\alpha \land \neg\beta)$	De Morgan
$(\alpha \land (\beta \lor \gamma)) \equiv ((\alpha \land \beta) \lor (\alpha \land \gamma))$	Distributivity of $\land$ over $\lor$
$(\alpha \lor (\beta \land \gamma)) \equiv ((\alpha \lor \beta) \land (\alpha \lor \gamma))$	Distributivity of $\lor$ over $\land$

Validity

- A sentence is *valid* if it is true in <u>all</u> models.
- Valid sentences are often called *tautologies*.
- Example:
    - $P \vee \neg P$
- Why is this useful?

Validity

- From validity we can derive the *deduction theorem*, which states that:
    - For any sentences α and β, $\alpha \models \beta$ if and only if the sentence $(\alpha \Rightarrow \beta)$ is valid
- Therefore we can describe *model checking* as checking if $(KB \Rightarrow \alpha)$ is valid!

Satisfiability

- A sentence is satisfiable if it is true in some model.
- If sentence α is true on a model m, we say that m satisfies α.

- Then, why is this useful?

Satisfiability

- We have a theorem that states:
    - $\alpha \vDash \beta$ if and only if the sentence $(\alpha \land \neg\beta)$ is unsatisfiable
- We can therefore prove $\alpha \vDash \beta$ by checking the unsatisfiability of $(\alpha \land \neg\beta)$
    - = false in all models
- It is a proof technique called *reductio ad absurdum* ("reduction to an absurd thing")
    - Or proof by refutation, or proof by contradiction

Inference rules

- We also have some useful inference rules.
- The two most important are *Modus Ponens* and *And-Elimination*.
- There are some other:
    - Modus Tollens
    - And-Addition
    - Or-Elimination
    - Or-Addition
    - …

Modus Ponens

- Definition:
  - If we have sentences of the form $(\alpha \Rightarrow \beta)$ and α is given, then β can be inferred.
- Written formally as:

$$\frac{\alpha \Rightarrow \beta, \quad \alpha}{\beta}$$

- Example:
  - (WumpusAhead ∧ WumpusAlive) ⇒ Shoot
  - If (WumpusAhead ∧ WumpusAlive) is given, Shoot can be inferred.

And-Elimination

- Definition:
  - If we have $(\alpha \land \beta)$ then α can be inferred.
- Written formally as:

$$\frac{\alpha \land \beta}{\alpha}$$

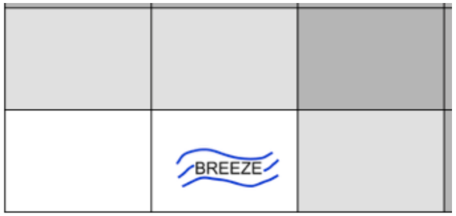
- Example:
  - (WumpusAhead $\land$ WumpusAlive)
  - Then WumpusAhead can be inferred.
  - Also works in the opposite direction, since $(\alpha \land \beta) \equiv (\beta \land \alpha)$

Now, what is all this good for?

Example

- KB:

R_1:	$\neg P_{(1,1)}$
R_2:	$B_{(1,1)} \Leftrightarrow (P_{(1,2)} \vee P_{(2,1)})$
R_3:	$B_{(2,1)} \Leftrightarrow (P_{(1,1)} \vee P_{(2,2)} \vee P_{(3,1)})$
R_4:	$\neg B_{(1,1)}$
R_5:	$B_{(2,1)}$



- Prove $\neg P_{(1,2)}$

PROOF FINDING

Example

Obtain new rule R_6 by biconditional elimination to R_2:

$$R_2: \quad \frac{B_{(1,1)} \Leftrightarrow (P_{(1,2)} \vee P_{(2,1)})}{(B_{(1,1)} \Rightarrow (P_{(1,2)} \vee P_{(2,1)})) \wedge ((P_{(1,2)} \vee P_{(2,1)}) \Rightarrow B_{(1,1)})} \quad :R_6$$

Apply And-Elimination to R_6 to obtain R_7:

$$R_6: \quad \frac{(B_{(1,1)} \Rightarrow (P_{(1,2)} \vee P_{(2,1)})) \wedge ((P_{(1,2)} \vee P_{(2,1)}) \Rightarrow B_{(1,1)})}{((P_{(1,2)} \vee P_{(2,1)}) \Rightarrow B_{(1,1)})} \quad :R_7$$

Contraposition on R_7 gives:

$$R_7: \quad \frac{((P_{(1,2)} \vee P_{(2,1)}) \Rightarrow B_{(1,1)})}{(\neg B_{(1,1)} \Rightarrow \neg(P_{(1,2)} \vee P_{(2,1)}))} \quad :R_8$$

PROOF FINDING

Example

Modus Ponens with R_8 and the fact R_4 gives:

R_8, R_4: $$\frac{(\neg B_{(1,1)} \Rightarrow \neg(P_{(1,2)} \vee P_{(2,1)})) \qquad \neg B_{(1,1)}}{\neg(P_{(1,2)} \vee P_{(2,1)})}$$

R_9: $\neg(P_{(1,2)} \vee P_{(2,1)})$

De Morgan's rule gives the final conclusion:

R_9: $$\frac{\neg(P_{(1,2)} \vee P_{(2,1)})}{\neg P_{(1,2)} \wedge \neg P_{(2,1)}}$$

R_{10}: $\neg P_{(1,2)} \wedge \neg P_{(2,1)}$

Conclusion: We have proved that there is no pit in (1,2) and no pit in (2,1).

PROOF FINDING

Proof finding

- Proof finding is a highly efficient inference algorithm since it can ignore irrelevant propositions.
- Several rules in the KB are not used, since they are not relevant for $P_{(1,2)}$.
- If we add a million sentences to KB it would not affect the performance of proving $P_{(1,2)}$, but it would lead to a huge performance drop for *model checking*.

Proof finding

- It is however not trivial to implement an algorithm for proof finding …
- … since we need to know which inference rule to apply when and where.

- A better option is the Resolution algorithm which only uses the resolution inference rule.

Resolution

- The Resolution inference rule combined with a <u>complete search algorithm</u> gives a complete inference algorithm.
- The Resolution rule:

R_{13}:	$\neg P_{(2,2)}$
R_{15}:	$P_{(1,1)} \vee P_{(2,2)} \vee P_{(3,1)}$

- The literal $\neg P_{(2,2)}$ resolves with the literal $P_{(2,2)}$ to give: $P_{(1,1)} \vee P_{(3,1)}$
    - If there is a pit in one of (1,1), (2,2) and (3,1) and it is not in (2,2), it must be in (1,1) or (3,1).

Resolution

- And if we continue resolve with R_1:

R_1:	$\neg P_{(1,1)}$
R_{16}:	$P_{(1,1)} \vee P_{(3,1)}$

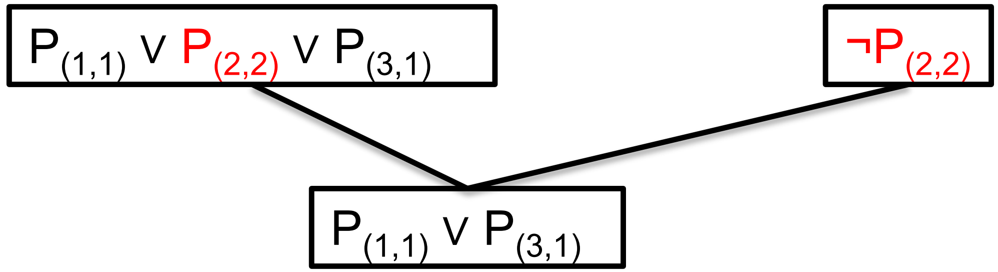
- $\neg P_{(1,1)}$ resolves with $P_{(1,1)}$ to give $P_{(3,1)}$
- We have proved that the pit is in (3,1).

Resolution

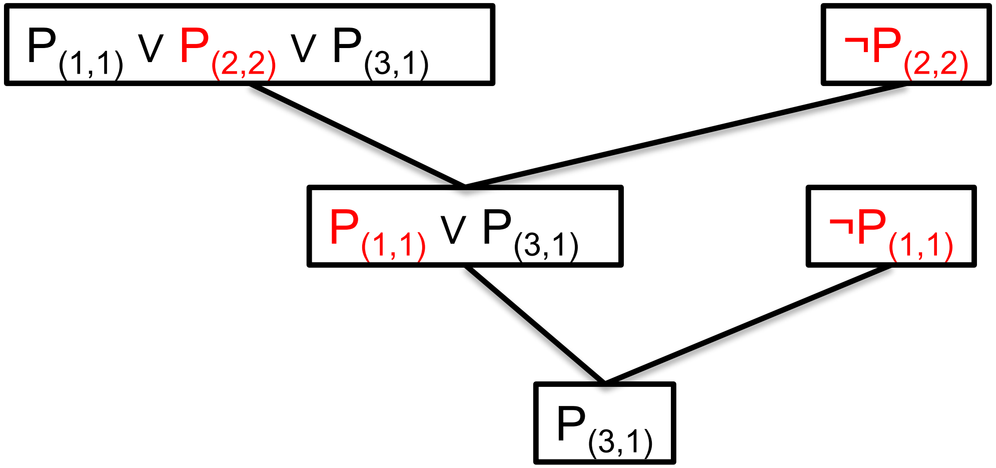
$P_{(1,1)} \vee P_{(2,2)} \vee P_{(3,1)}$

$\neg P_{(2,2)}$

Resolution



Resolution



Resolution

- Problem: The resolution rule only applies to disjunctions of literals.
- Solution: ”*Every sentence of propositional logic is logically equivalent to a conjunction of disjunctions of literals*”

- A sentence in this form is said to be in Conjunctive Normal Form (CNF).

CNF example

- Task: Convert $B_{(1,1)} \Leftrightarrow (P_{(1,2)} \vee P_{(2,1)})$ to CNF.
- Solution:

  1. Eliminate $\Leftrightarrow$ with biconditional elimination to get:
     $(B_{(1,1)} \Rightarrow (P_{(1,2)} \vee P_{(2,1)})) \wedge ((P_{(1,2)} \vee P_{(2,1)}) \Rightarrow B_{(1,1)})$
  2. Eliminate $\Rightarrow$ with implication elimination to get:
     $(\neg B_{(1,1)} \vee P_{(1,2)} \vee P_{(2,1)}) \wedge (\neg(P_{(1,2)} \vee P_{(2,1)}) \vee B_{(1,1)})$
  3. Move $\neg$ inwards with De Morgan and/or double-negation elimination:
     $(\neg B_{(1,1)} \vee P_{(1,2)} \vee P_{(2,1)}) \wedge ((\neg P_{(1,2)} \wedge \neg P_{(2,1)}) \vee B_{(1,1)})$
  4. Apply distributivity laws to get:
     $(\neg B_{(1,1)} \vee P_{(1,2)} \vee P_{(2,1)}) \wedge (\neg P_{(1,2)} \vee B_{(1,1)}) \wedge (\neg P_{(2,1)} \vee B_{(1,1)})$

     … which is a conjunction of three disjunctions.

The Resolution algorithm

- Prove KB $\models \alpha$ by showing that (KB $\wedge \neg\alpha$) is unsatisfiable.
- Steps:
  1. Convert (KB $\wedge \neg\alpha$) to CNF.
  2. Apply resolution rule to the resulting clauses.
  3. Apply resolution rule to, if any, pairs that contain complementary literals.
  4. Continue until one of two things happen:
     - No new clauses can be added, in which case KB does not entail α.
     - Two clauses resolve to the empty clause, in which case KB entails α. Example: $P_{(1,2)}$ and $\neg P_{(1,2)}$ resolves to $\square$

A special case

- If the knowledge base only contains *Horn clauses*, we can use a simpler resolution inference algorithm.
- Horn clause: A disjunction of literals of which <u>at most one is positive</u>.
- Seems overly restrictive, but is actually quite common in many real-world examples.

Horn clauses

- The restriction is important for three reasons:
  - If we have
    $(\neg L_{(1,1)} \vee \neg Breeze \vee B_{(1,1)})$
    and we know it is a Horn clause, we can rewrite it as the implication:
    $(L_{(1,1)} \wedge Breeze) \Rightarrow B_{(1,1)}$
    which can be read as “if the player L is in (1,1) and there is a breeze, then (1,1) is breezy”. It is much easier and more intuitive for humans to read.

Horn clauses

  - Inference with Horn clauses can be done with the efficient *forward chaining* and *backward chaining* algorithms.
  - The computation can be done in time that is linear in the size of KB.

- Horn clauses with exactly one positive literal are called *definite clauses*.
- Definite clauses asserts a given proposition – a *fact*.

Forward chaining

- FC determines if a single proposition symbol q is entailed by a KB of Horn clauses.
- Steps:
  - Start with the facts (positive literals) in KB.
  - If all premises of an implication is known, we can add the conclusion to KB.
  - Continue until no more inferences can be made, or q is added to KB.

Forward chaining

- Example:

F_1:	$L_{(1,1)}$
F_2:	Breeze
R_1:	$(L_{(1,1)} \wedge \text{Breeze}) \Rightarrow B_{(1,1)}$

- Both premises $L_{(1,1)}$ and Breeze in R_1 are known.
- We can therefore add $B_{(1,1)}$ to KB to get:

F_1:	$L_{(1,1)}$
F_2:	Breeze
F_3:	$B_{(1,1)}$
R_1:	$(L_{(1,1)} \wedge \text{Breeze}) \Rightarrow B_{(1,1)}$

If q is $B_{(1,1)}$ we have proved that q is entailed by KB.

Forward chaining

- Forward chaining is *sound* and *complete*, and most importantly runs in linear time.
- It is a data-driven reasoning approach: it starts with the known facts.
- The most intuitive way of understanding forward chaining is by an AND-OR graph.

AND-OR graph

KB
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B

Prove Q

Start with the facts

A

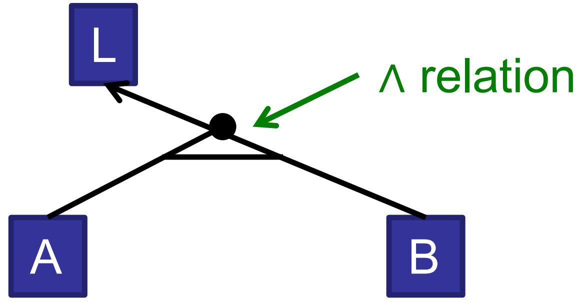
B

AND-OR graph

KB
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B

Prove Q

From A and B we can add L

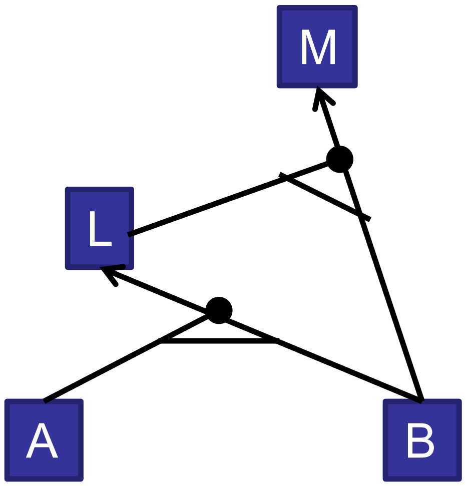


AND-OR graph

KB
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B

Prove Q

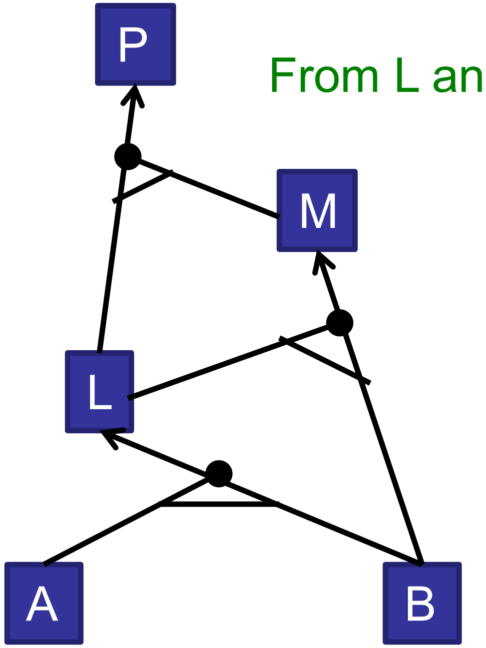
From B and L we can add M



AND-OR graph

KB
P ⇒ Q
L ∧ M ⇒ P
B ∧ L ⇒ M
A ∧ P ⇒ L
A ∧ B ⇒ L
A
B

Prove Q



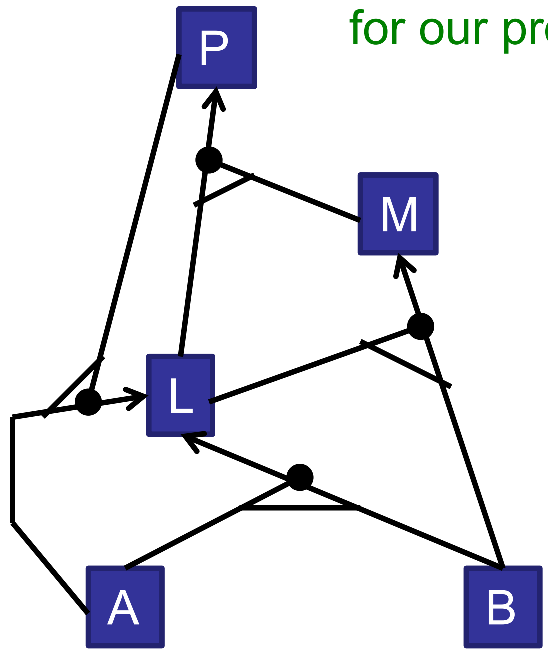
From L and M we can add P

AND-OR graph

KB
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B

Prove Q

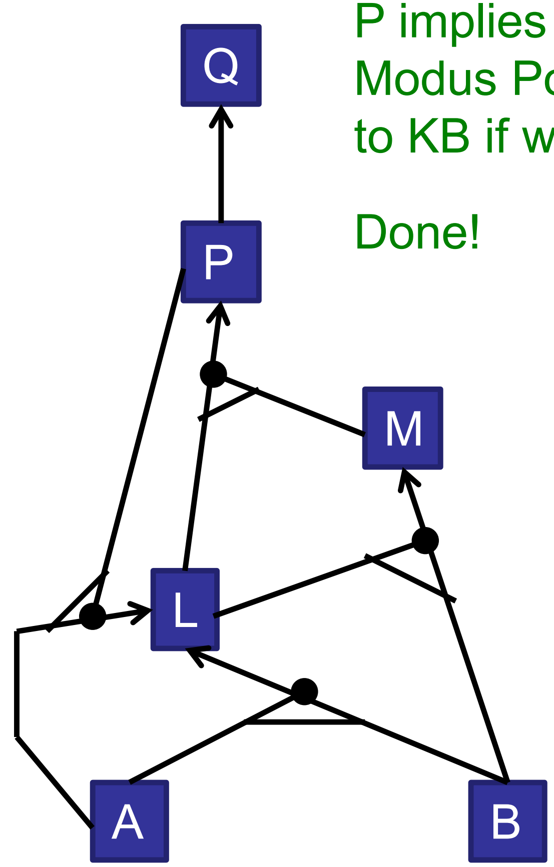
L can also be implied from
A and P (but it is not necessary
for our proof).



AND-OR graph

KB
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B

Prove Q



P implies Q. According to Modus Ponens we can add Q to KB if we have proved P.

Done!

Backward chaining

- BC is a goal-directed reasoning.
- It is useful to answer questions such as:
    - What shall I do now?
    - Where are my keys?
- BC is very similar to FC, but work backwards from the query to find the facts needed to answer q.
- BC often runs in less than linear time.
- An efficient AI should use both forward and backward chaining, and use the most appropriate for each question.

AND-OR graph

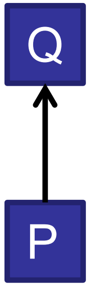
KB
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B

Q

Start with the Query.

AND-OR graph

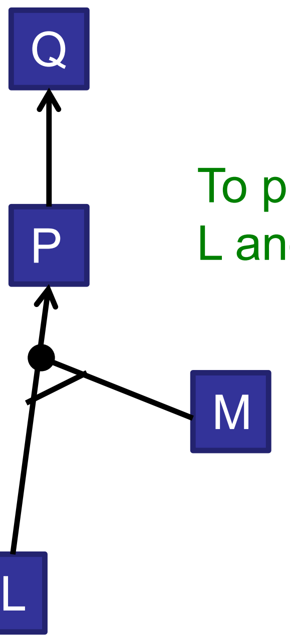
KB
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B



To prove Q we need to prove P.

AND-OR graph

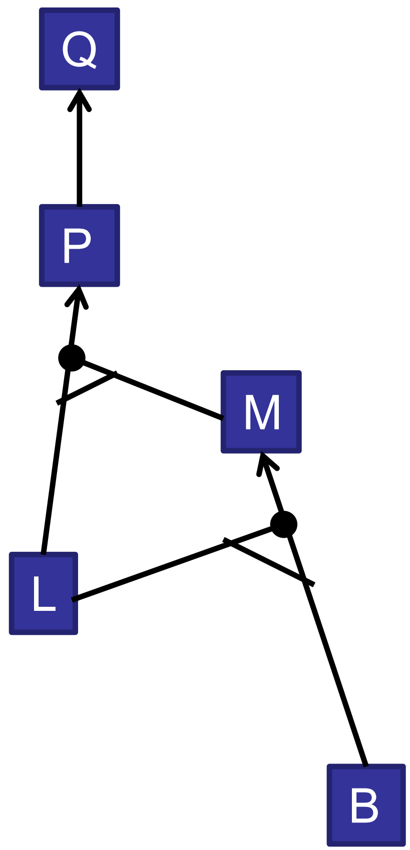
KB
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B



To prove P we need to prove L and M.

AND-OR graph

KB
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B

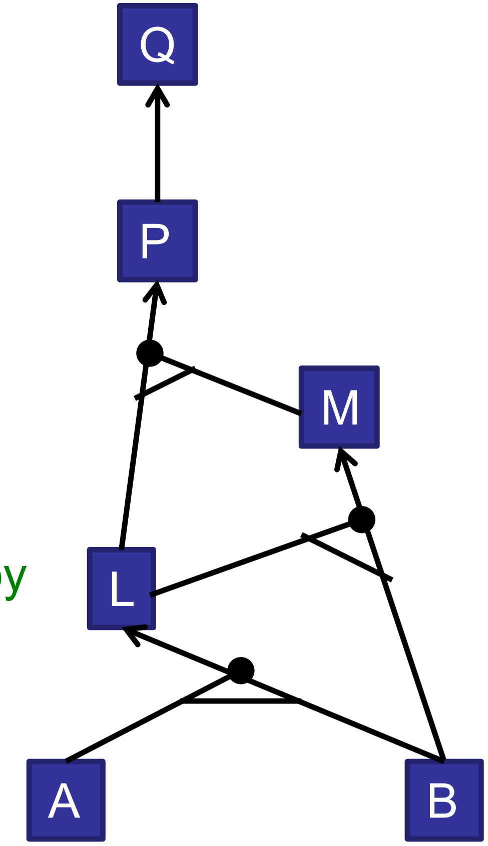


M is proved
by B and L.

AND-OR graph

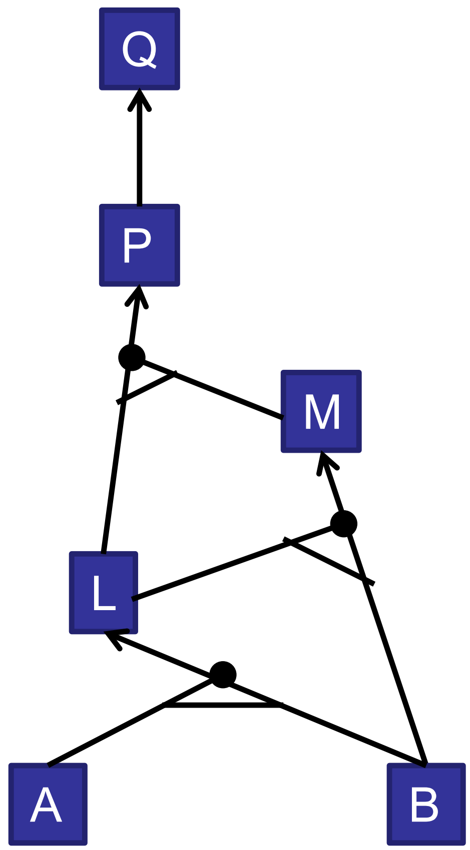
KB
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B

L is proved by
A and B.



AND-OR graph

KB
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B



Since A and B is known,
we have proved Q.

Summary

- With inference algorithms we can solve many logic problems.
  - Model checking
  - Forward- and Backward chaining
- One problem is still performance since all has a worst case scenario with exponential time complexity.
- There are other issues as well, which we will discuss in the next lecture.

That was all for this lecture

Acknowledgements

Dr. Johan Hagelbäck

Linnæus University

johan.hagelback@lnu.se

http://aiguy.org