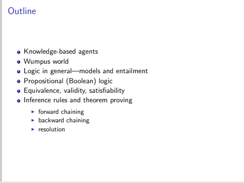
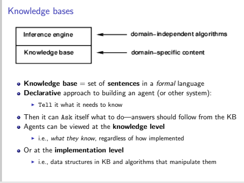
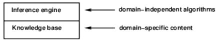


Outline

Today we will focus on knowledge, how can we manage it, store and retrieve it, and how can we conclude new information from the old. How can we teach it to an agent? To see the challenges we use a simple game: the Wumpus world. Next we give a short session about logic, the main concepts, and its mechanicasition.



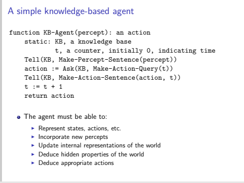
Knowledge bases



- **Knowledge base** = set of **sentences** in a *formal* language
- **Declarative** approach to building an agent (or other system):
  - `Tell` it what it needs to know
- Then it can `Ask` itself what to do—answers should follow from the KB
- Agents can be viewed at the **knowledge level**
  - i.e., *what they know*, regardless of how implemented
- Or at the **implementation level**
  - i.e., data structures in KB and algorithms that manipulate them

For us a knowledge base is just a **set** of formulae, the statements are given in a specific artificial (formal) language. As it is a set, we can broaden this base by adding new formulae, i.e. we tell the new formulae to the agent/knowledge base. Similarly we can narrow this base by deleting formulae, but it is easier if the agent drops any old formulae that contradict the new knowledge. The agent can use this knowledge base by **asking** it about the next action.

We can treat the knowledge base and its inference engine (which knows the logic) as a *black box*, we tell it the facts and rules, make queries from it, while we have no interest in its internal structure. But you can treat the KB and IE as a programming problem: what data structures are needed for efficient knowledge handling, and what algorithms are needed to implement the ability to suggest the next action.



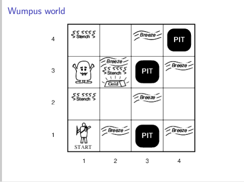
A simple knowledge-based agent

```
function KB-Agent(percept): an action
    static: KB, a knowledge base
            t, a counter, initially 0, indicating time
    Tell(KB, Make-Percept-Sentence(percept))
    action := Ask(KB, Make-Action-Query(t))
    Tell(KB, Make-Action-Sentence(action, t))
    t := t + 1
    return action
```

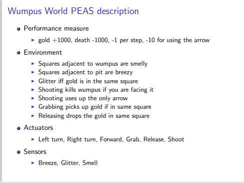
- The agent must be able to:
  - Represent states, actions, etc.
  - Incorporate new percepts
  - Update internal representations of the world
  - Deduce hidden properties of the world
  - Deduce appropriate actions

From a birds-eye view the program of a knowledge agent is very simple. As an input it needs the actual perception, and its memory contains the KB and a timer. It adds the actual perception (as a set of formulae) to the KB. Next it asks the KB about the next action (based on the time), and tells the KB that the agent is executing this action now. The time updates, and the next action becomes the output.

We see from the list below, what complexity is hidden in the function calls in the listing.


Wumpus world

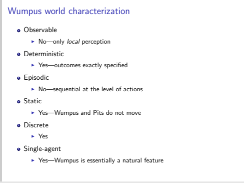
Let take the this game, where you need to find the gold, and avoid from the monster called Wumpus. You are in a dark maze, and there are traps everywhere. Fortunately, you can recognize these based on a breeze and the smell of the monster.



Wumpus World PEAS description

- Performance measure
  - gold +1000, death -1000, -1 per step, -10 for using the arrow
- Environment
  - Squares adjacent to wumpus are smelly
  - Squares adjacent to pit are breezy
  - Glitter iff gold is in the same square
  - Shooting kills wumpus if you are facing it
  - Shooting uses up the only arrow
  - Grabbing picks up gold if in same square
  - Releasing drops the gold in same square
- Actuators
  - Left turn, Right turn, Forward, Grab, Release, Shoot
- Sensors
  - Breeze, Glitter, Smell

If we want to compare the different agents that want to acquire the treasure, we need some measurement to compare them. We can use the well-known measure of performance, so we need to score each action. The rules of the game are detailed here. To give a complete description we need to list all the possible actions and the different kinds of percepts.

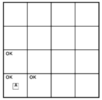


Wumpus world characterization

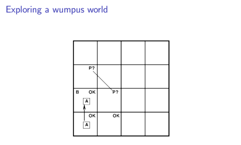
- Observable
  - No—only *local* perception
- Deterministic
  - Yes—outcomes exactly specified
- Episodic
  - No—sequential at the level of actions
- Static
  - Yes—Wumpus and Pits do not move
- Discrete
  - Yes
- Single-agent
  - Yes—Wumpus is essentially a natural feature

We have seen at the beginning of the semester, the different types of the environments given different kinds of challenges. Lets see the exact type for this game! We can see that this is a simple problem, just the partial observability and episodic nature cause some difficulties.

Exploring a wumpus world

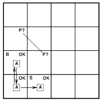


Let us assume we start the adventure at the bottom left, and in this room there is no trap, so we alive at the beginning. We do not smell anything, and feel no breeze in here, so in the neighbouring squares there is no Wumpus and no trap. It is safe to discover some adjacent room.

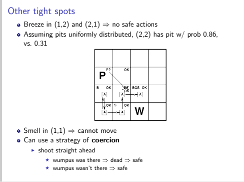


In the next room we feel the breeze, so in some (or more) adjacent rooms have a trap. So it is not safe to move forward, therefore we go back, and try the other direction, but there we catch the smell of the Wumpus. What do we do now? If we know the logic, we can prove that the trap (pit) must be in a unique room, so the other room is safe, you we continue our adventure.

Exploring a wumpus world

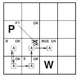


As all neighbours are safe, we can discover the unknown rooms and find the gold.



Other tight spots

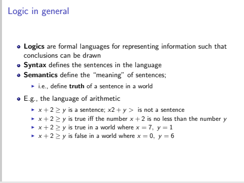
- Breeze in (1,2) and (2,1) ⇒ no safe actions
- Assuming pits uniformly distributed, (2,2) has pit w/ prob 0.86, vs. 0.31



- Smell in (1,1) ⇒ cannot move
- Can use a strategy of **coercion**
  - shoot straight ahead
    - wumpus was there ⇒ dead ⇒ safe
    - wumpus wasn't there ⇒ safe

There are cases when logic is not enough, we cannot prove anything. If we feel breeze in both rooms neighbouring the starting square, we cannot deduce the position of pit(s). The probabilistic reasoning could help in this case, so that we need to avoid the middle of the labyrinth (but it does not guarantee anything.)

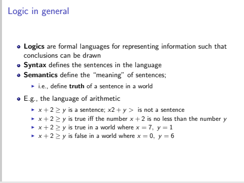
If we fell a breeze or a smell at the starting point, we cannot move safely. But if we feel a smell only, we can use our arrow. If the Wumpus is in some direction, by shooting that way we kill it, so this direction now becomes safe. Otherwise it was safe before, anyway. So now we can go in that direction.



Logic in general

- **Logics** are formal languages for representing information such that conclusions can be drawn
- **Syntax** defines the sentences in the language
- **Semantics** define the "meaning" of sentences;
  - i.e., define **truth** of a sentence in a world
- E.g., the language of arithmetic
  - $x + 2 \geq y$ is a sentence; $x2 + y >$ is not a sentence
  - $x + 2 \geq y$ is true iff the number $x + 2$ is no less than the number y
  - $x + 2 \geq y$ is true in a world where $x = 7,\ y = 1$
  - $x + 2 \geq y$ is false in a world where $x = 0,\ y = 6$

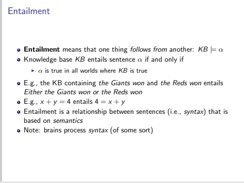
Now let us refresh our knowledge about logic. There are different kinds of logics, you have learned about two: zero-order and first-order. In the following we will use the zero-order approach for this toy-problem, but for real life problems the first-order approach gives simpler/shorter formulae, we can handle problems in a more compact way. The syntax is the grammar of the logic, or other artificial languages. It contains the rules of what is a well formed formula. GIven a syntax, we can construct a parser which gives back the structure of the formula. The syntax of the arithmetic gives that the first example is a real statement, but the second ($x2 + y >$) is not.



Logic in general

- **Logics** are formal languages for representing information such that conclusions can be drawn
- **Syntax** defines the sentences in the language
- **Semantics** define the “meaning” of sentences;
  - i.e., define **truth** of a sentence in a world
- E.g., the language of arithmetic
  - $x + 2 \geq y$ is a sentence; $x2 + y >$ is not a sentence
  - $x + 2 \geq y$ is true iff the number $x + 2$ is no less than the number y
  - $x + 2 \geq y$ is true in a world where $x = 7,\ y = 1$
  - $x + 2 \geq y$ is false in a world where $x = 0,\ y = 6$

The grammar gives meaning to the formula. Usually we do it in a recursive way. In case of arithmetics we need to give meaning to $+$ and $\geq$. The standard interpretation assigns the function addition and the relation less-or-equal to these signs. Moreover the arithmetic is about numbers. What do we need to understand about these numbers? Are they natural numbers, integers, rational fractions, real or complex numbers? The domain of the variables is part of the definition of the interpretation. As we use variables in arithmetics, and their values is not fixed, they can change, so there is no reason to fix these values in the interpretation. But these values can create a valuation together. This is the reason why an interpretation and a valuation together gives a model in first-order logic.



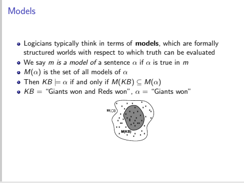
Entailment

- **Entailment** means that one thing *follows from* another: $KB \models \alpha$
- Knowledge base KB entails sentence α if and only if
  - α is true in all worlds where KB is true
- E.g., the KB containing *the Giants won* and *the Reds won* entails *Either the Giants won or the Reds won*
- E.g., $x + y = 4$ entails $4 = x + y$
- Entailment is a relationship between sentences (i.e., *syntax*) that is based on *semantics*
- Note: brains process *syntax* (of some sort)

We can treat the word entailment as a synonym of logical consequence. Maybe you remember its definition, and the property given here. There is a hidden implication in this property, and not an equivalence. If KB is true in some world (understand this as in a model, or an interpretation), then formula alpha must be true. If KB is false in some world, then that world is not interesting for us, although formula alpha can be true here. The entailment does not hold if there exists a world, where KB is true and formula alpha is false.

Here you can find some cases where the entailment holds.

In general, the formula alpha and formulae in KB are based on syntax. The concept of logical consequence belongs to semantics. We need to check that in all model of KB whether the formula alpha is true. In zero-order case we have used truth-table for this. In first-order logic usually a KB has infinitely many models, so we cannot list all of them.

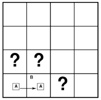


Models

- Logicians typically think in terms of **models**, which are formally structured worlds with respect to which truth can be evaluated
- We say *m is a model of* a sentence α if α is true in m
- $M(\alpha)$ is the set of all models of α
- Then $KB \models \alpha$ if and only if $M(KB) \subseteq M(\alpha)$
- $KB =$ “Giants won and Reds won”, $\alpha =$ “Giants won”

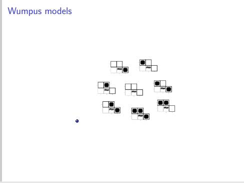
In the following we will use zero-order logic, so concepts model and interpretation overlap. We can take the set of interpretations where a formula (or a set of formulae) is true, i.e. the set of its (or their) models. We can draw up the entailment as a subset property between the models.

Entailment in the wumpus world

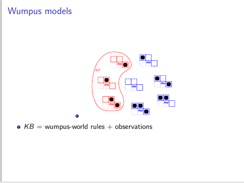
- Situation after detecting nothing in [1,1],
- moving right, breeze in [2,1]
- Consider possible models for ?s assuming only pits
- 

- 3 Boolean choices ⇒ 8 possible models

As we refresh the theory, lets see them in practice. In our example the starting position is safe, and in the adjacent room we feel a breeze. What can we say about the rooms nearby, which way can we continue our journey? (They are denoted with a question mark.) Can you conclude something alone now?



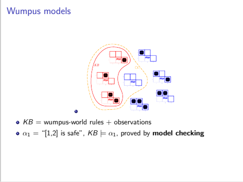
We have three question marks, we can use three boolean variables to describe the situation. There is either a trap or not in one given room, so these are two options; and we have 3 rooms, so it gives 222=8 cases. You can see all the cases here.



Wumpus models

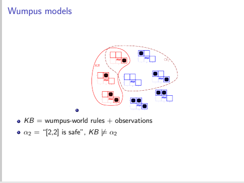
- KB = wumpus-world rules + observations

We coloured cases that comply with the rules of this game red.


Wumpus models

- KB = wumpus-world rules + observations
- α_1 = "[1,2] is safe", $KB \models \alpha_1$, proved by **model checking**

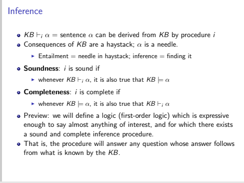
If the formula α_1 denotes that the room above the starting position is safe, its model has four elements. $M(KB)$ is a subset of $M(\alpha)$, so the entailment holds. We solved this question by checking the models.



Wumpus models

- KB = wumpus-world rules + observations
- α_2 = "[2,2] is safe", $KB \not\models \alpha_2$

We can ask whether the room next to previous one is safe (α_2). There are four models of this statement, and we can see, that $M(KB)$ is not a subset of $M(\alpha_2)$ there are interpretations, where KB is true and formula alpha is false, so the entailment does not hold.

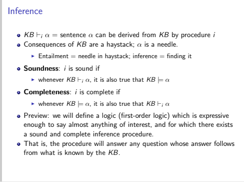


Inference

- $KB \vdash_i \alpha$ = sentence α can be derived from KB by procedure i
- Consequences of KB are a haystack; α is a needle.
  - Entailment = needle in haystack; inference = finding it
- **Soundness**: i is sound if
  - whenever $KB \vdash_i \alpha$, it is also true that $KB \models \alpha$
- **Completeness**: i is complete if
  - whenever $KB \models \alpha$, it is also true that $KB \vdash_i \alpha$
- Preview: we will define a logic (first-order logic) which is expressive enough to say almost anything of interest, and for which there exists a sound and complete inference procedure.
- That is, the procedure will answer any question whose answer follows from what is known by the KB.

The last 30 years produced many results for solving SAT problems, which are tightly connected to entailment, so the situation is actually better than described in the AIMA book.

The logical consequence (entailment, model checking) belong to semantics. In logic there is a concept of syntactic consequence, inference or calculus, which belongs to the syntax. We can imagine the situation such that if we treat KB as a haystack, then the formula alpha is the needle. At entailment we know that there is a needle in the haystack. At inference we need to construct the proof, i.e. we need to find the needle in the haystack.

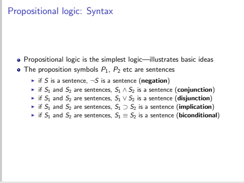


Inference

- $KB \vdash_i \alpha$ = sentence α can be derived from KB by procedure i
- Consequences of KB are a haystack; α is a needle.
  - Entailment = needle in haystack; inference = finding it
- **Soundness**: i is sound if
  - whenever $KB \vdash_i \alpha$, it is also true that $KB \models \alpha$
- **Completeness**: i is complete if
  - whenever $KB \models \alpha$, it is also true that $KB \vdash_i \alpha$
- Preview: we will define a logic (first-order logic) which is expressive enough to say almost anything of interest, and for which there exists a sound and complete inference procedure.
- That is, the procedure will answer any question whose answer follows from what is known by the KB.

The connection between entailment and inference is a serious question. An inference is sound, when if you prove something, then it is true. (If we construct the inference rules carefully, then the calculus will be automatically be sound.) The reverse direction, when something is true, then we can prove it (that is, we can construct a proof for it) is much harder. But for zero-order logic we have a complete method, and so do we for some specific first-order logic

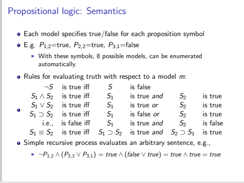
We prefer the sound and complete calculus.



Propositional logic: Syntax

- Propositional logic is the simplest logic—illustrates basic ideas
- The proposition symbols P_1, P_2 etc are sentences
  - if S is a sentence, $\neg S$ is a sentence (**negation**)
  - if S_1 and S_2 are sentences, $S_1 \wedge S_2$ is a sentence (**conjunction**)
  - if S_1 and S_2 are sentences, $S_1 \vee S_2$ is a sentence (**disjunction**)
  - if S_1 and S_2 are sentences, $S_1 \supset S_2$ is a sentence (**implication**)
  - if S_1 and S_2 are sentences, $S_1 \equiv S_2$ is a sentence (**biconditional**)

In the introductory logic course we have used this notation.



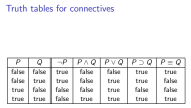
Propositional logic: Semantics

- Each model specifies true/false for each proposition symbol
- E.g. $P_{1,2}$=true, $P_{2,2}$=true, $P_{3,1}$=false
  - With these symbols, 8 possible models, can be enumerated automatically.
- Rules for evaluating truth with respect to a model m:
-

$\neg S$	is true iff	S	is false		
$S_1 \wedge S_2$	is true iff	S_1	is true *and*	S_2	is true
$S_1 \vee S_2$	is true iff	S_1	is true *or*	S_2	is true
$S_1 \supset S_2$	is true iff	S_1	is false *or*	S_2	is true
i.e.,	is false iff	S_1	is true *and*	S_2	is false
$S_1 \equiv S_2$	is true iff	$S_1 \supset S_2$	is true *and*	$S_2 \supset S_1$	is true

- Simple recursive process evaluates an arbitrary sentence, e.g.,
  - $\neg P_{1,2} \wedge (P_{2,2} \vee P_{3,1}) = \mathit{true} \wedge (\mathit{false} \vee \mathit{true}) = \mathit{true} \wedge \mathit{true} = \mathit{true}$

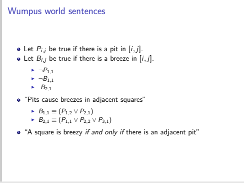
The rules of the semantics are the same, we just use 0 and 1 instead of *false* and *true*.



Truth tables for connectives

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \supset Q$	$P \equiv Q$
false	false	true	false	false	true	true
false	true	true	false	true	true	false
true	false	false	false	true	false	false
true	true	false	true	true	true	true

This is same, we just use 0 and 1 instead of *false* and *true*.

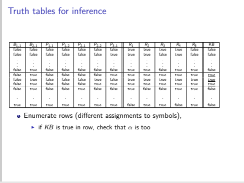


Wumpus world sentences

- Let $P_{i,j}$ be true if there is a pit in $[i,j]$.
- Let $B_{i,j}$ be true if there is a breeze in $[i,j]$.
  - $\neg P_{1,1}$
  - $\neg B_{1,1}$
  - $B_{2,1}$
- "Pits cause breezes in adjacent squares"
  - $B_{1,1} \equiv (P_{1,2} \vee P_{2,1})$
  - $B_{2,1} \equiv (P_{1,1} \vee P_{2,2} \vee P_{3,1})$
- "A square is breezy *if and only if* there is an adjacent pit"

So let us see logic in action. We introduce variables to denote whether a room has a trap/pit; and whether in a room there is a breeze. Using these variables we can write (with variables only), that we did not feel anything at the starting position, and in the next room there is a breeze.

You may formulate the rule according to the traps, that if in a room there is a trap, then in all the adjacent rooms there is a breeze. Now it is better to use a different point of view: if there is a breeze in the room, then in some adjacent room there is a trap. But this is not enough, we need to add that if in a room there is no breeze, then in all adjacent rooms there is no trap. If you use de Morgan rule, you can join the two implications into a equivalence (biconditional). Let us denote these 5 formula here with $R_1,\ldots,R_5$!

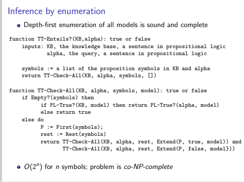


Truth tables for inference

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	R_1	R_2	R_3	R_4	R_5	KB
false	false	false	false	false	false	false	true	true	true	true	false	false
false	false	false	false	false	false	true	true	true	false	true	false	false
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
false	true	false	false	false	false	false	true	true	false	true	true	false
false	true	false	false	false	false	true	true	true	true	true	true	<u>*true*</u>
false	true	false	false	false	true	false	true	true	true	true	true	<u>*true*</u>
false	true	false	false	false	true	true	true	true	true	true	true	<u>*true*</u>
false	true	false	false	true	false	false	true	false	false	true	true	false
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
true	true	true	true	true	true	true	false	true	true	false	true	false

- Enumerate rows (different assignments to symbols),
  - if *KB* is true in row, check that α is too

We have 7 variables, so we need a truth table with 128 rows. We have columns for R_1,...,R_5 and KB (which is the conjunction of R_1,...,R_5). We are interested in the rows, where KB is true. 3 such rows are exist, and we need to check whether formula α is true here (for any formula α).



Inference by enumeration

- Depth-first enumeration of all models is sound and complete

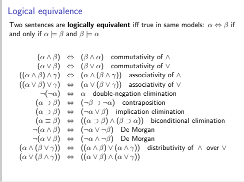
```
function TT-Entails?(KB,alpha): true or false
    inputs: KB, the knowledge base, a sentence in propositional logic
            alpha, the query, a sentence in propositional logic

    symbols := a list of the proposition symbols in KB and alpha
    return TT-Check-All(KB, alpha, symbols, [])

function TT-Check-All(KB, alpha, symbols, model): true or false
    if Empty?(symbols) then
          if PL-True?(KB, model) then return PL-True?(alpha, model)
          else return true
    else do
          P := First(symbols);
          rest := Rest(symbols)
          return TT-Check-All(KB, alpha, rest, Extend(P, true, model)) and
                 TT-Check-All(KB, alpha, rest, Extend(P, false, model}))
```

- $O(2^n)$ for n symbols; problem is *co-NP-complete*

If we use a computer, we do not need to construct a traditional truth-table. We can list all the interpretations in a recursive way. It looks like a DFS, where all the leaves are at level n. If we reach a leaf of this tree, we need to check whether this interpretation is model of KB. If not, go back. Otherwise we need to check whether it is a model of formula alpha. If not, we found a counter example, and we can stop the whole process. Otherwise we need to discover the remaining parts of the tree. This tree has 2^n leaves, and in the worst case we need to visit all of them.

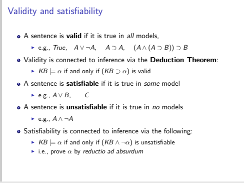


Logical equivalence

Two sentences are **logically equivalent** iff true in same models: $\alpha \Leftrightarrow \beta$ if and only if $\alpha \models \beta$ and $\beta \models \alpha$

$$\begin{array}{rcll}
(\alpha \land \beta) & \Leftrightarrow & (\beta \land \alpha) & \text{commutativity of } \land \\
(\alpha \lor \beta) & \Leftrightarrow & (\beta \lor \alpha) & \text{commutativity of } \lor \\
((\alpha \land \beta) \land \gamma) & \Leftrightarrow & (\alpha \land (\beta \land \gamma)) & \text{associativity of } \land \\
((\alpha \lor \beta) \lor \gamma) & \Leftrightarrow & (\alpha \lor (\beta \lor \gamma)) & \text{associativity of } \lor \\
\neg(\neg\alpha) & \Leftrightarrow & \alpha & \text{double-negation elimination} \\
(\alpha \supset \beta) & \Leftrightarrow & (\neg\beta \supset \neg\alpha) & \text{contraposition} \\
(\alpha \supset \beta) & \Leftrightarrow & (\neg\alpha \lor \beta) & \text{implication elimination} \\
(\alpha \equiv \beta) & \Leftrightarrow & ((\alpha \supset \beta) \land (\beta \supset \alpha)) & \text{biconditional elimination} \\
\neg(\alpha \land \beta) & \Leftrightarrow & (\neg\alpha \lor \neg\beta) & \text{De Morgan} \\
\neg(\alpha \lor \beta) & \Leftrightarrow & (\neg\alpha \land \neg\beta) & \text{De Morgan} \\
(\alpha \land (\beta \lor \gamma)) & \Leftrightarrow & ((\alpha \land \beta) \lor (\alpha \land \gamma)) & \text{distributivity of } \land \text{ over } \lor \\
(\alpha \lor (\beta \land \gamma)) & \Leftrightarrow & ((\alpha \lor \beta) \land (\alpha \lor \gamma)) &
\end{array}$$

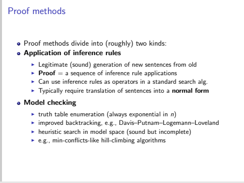
We want to show a slightly better method, which was great to implement in the seventies and thus construct a new (logical) programming language. However, it needs a long path. The logical equivalence is a known concept from the introductory logic course, and we learned how to rewrite most of these rules.



Validity and satisfiability

- A sentence is **valid** if it is true in *all* models,
  - e.g., *True*, $A \lor \neg A$, $A \supset A$, $(A \land (A \supset B)) \supset B$
- Validity is connected to inference via the **Deduction Theorem**:
  - $KB \models \alpha$ if and only if $(KB \supset \alpha)$ is valid
- A sentence is **satisfiable** if it is true in *some* model
  - e.g., $A \lor B$, C
- A sentence is **unsatisfiable** if it is true in *no* models
  - e.g., $A \land \neg A$
- Satisfiability is connected to inference via the following:
  - $KB \models \alpha$ if and only if $(KB \land \neg\alpha)$ is unsatisfiable
  - i.e., prove α by *reductio ad absurdum*

Similarly we have looked at the concepts of satisfiable, unsatisfiable and valid formulae. The definition was different, but the listed properties hold. The last lines gives our definition as a property, and this will be used in the following.



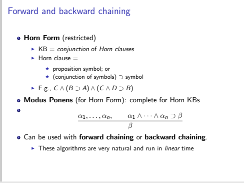
Proof methods

- Proof methods divide into (roughly) two kinds:
- **Application of inference rules**
  - Legitimate (sound) generation of new sentences from old
  - **Proof** = a sequence of inference rule applications
  - Can use inference rules as operators in a standard search alg.
  - Typically require translation of sentences into a **normal form**
- **Model checking**
  - truth table enumeration (always exponential in n)
  - improved backtracking, e.g., Davis–Putnam–Logemann–Loveland
  - heuristic search in model space (sound but incomplete)
  - e.g., min-conflicts-like hill-climbing algorithms

There are two ways to prove (logical or syntactic) consequences. The first is based on syntax, it uses inference rules and constructs a sequence of formulae (sometimes in tree format). As we have a limited number of inference rules we can apply some search method. To reduce the number of inference rules we rewrite the original formulae into normal forms.

The other way is the model checking. We do not need to generate all the interpretations, the DPLL method from 1963 works well, and this method was improved in several way (Wikipedia Boolean Satisfiability problem).

The local search can be used to solve such problems. Unfortunately it is not a complete method. (GSAT, WalkSAT)



Forward and backward chaining

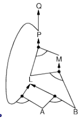
- **Horn Form** (restricted)
  - KB = *conjunction* of *Horn clauses*
  - Horn clause =
    - proposition symbol; or
    - (conjunction of symbols) $\supset$ symbol
  - E.g., $C \wedge (B \supset A) \wedge (C \wedge D \supset B)$
- **Modus Ponens** (for Horn Form): complete for Horn KBs
- $$\frac{\alpha_1, \ldots, \alpha_n, \qquad \alpha_1 \wedge \cdots \wedge \alpha_n \supset \beta}{\beta}$$
- Can be used with **forward chaining** or **backward chaining**.
  - These algorithms are very natural and run in *linear* time

If we take a specific logic, we can get an effective method. This is really interesting in first-order logic, but it takes several lectures to understand every detail. Therefore we show the essence in zero-order logic. To use these methods we need to rewrite our formula into conjunctive normal form. But not all CNF formulae are good for us. We need all elementary disjunctions (or clause) to contains at least one positive literal. If the whole clause is that literal, i.e. propositional variable, we call it a fact. If it contains exactly one positive literal, then it can be rewritten into implication form, where there is no negation. We can use the MP as the only inference rule. This is enough for us, as this will be a complete calculus for Horn clauses.

The question is: how will we use this rule? Two different approaches give two methods: forward and backward chaining. These methods have linear complexity.

Forward chaining

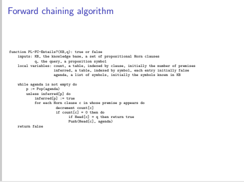
- Idea: fire any rule whose premises are satisfied in the *KB*, add its conclusion to the *KB*, until query is found



- $P \supset Q$, $L \wedge M \supset P$, $B \wedge L \supset M$, $A \wedge P \supset L$, $A \wedge B \supset L$, A, B

In forward chaining we attack at full width, if there is an implication whose premises are true (occurs in KB), then we add the suffix of the implication to the KB too. We repeat this process until the query is found, or the rule is not applicable to get any new formulae.

To understand these methods, we take this set of Horn clauses, and visualise them as a directed graph.



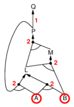
Forward chaining algorithm

```
function PL-FC-Entails?(KB,q): true or false
    inputs: KB, the knowledge base, a set of propositional Horn clauses
            q, the query, a proposition symbol
    local variables: count, a table, indexed by clause, initially the number of premises
                     inferred, a table, indexed by symbol, each entry initially false
                     agenda, a list of symbols, initially the symbols known in KB

    while agenda is not empty do
        p := Pop(agenda)
        unless inferred[p] do
            inferred[p] := true
            for each Horn clause c in whose premise p appears do
                      decrement count[c]
                      if count[c] = 0 then do
                            if Head[c] = q then return true
                            Push(Head[c], agenda)
    return false
```

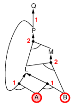
If we store and regularly update how many premises of a clause is false/unknown, and sort them based on this number, we can easily determine the next implication to apply the rule MP to. If we cannot apply MP again to get something new, we stop.

Forward chaining example



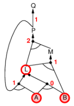
In the beginning we have two facts, A and B. The numbers at the implications denote the number of their premises.

Forward chaining example



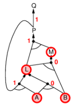
As we selected fact A, we decrease the counter of implications containing A as a hypothesis.

Forward chaining example



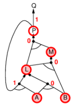
Next we select B, and decrease the counters related to B. Now one counter becomes zero ($A \wedge B \supset L$), so the consequence L is added to the agenda.

Forward chaining example

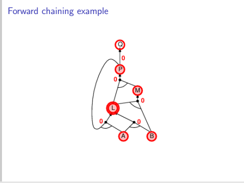


Now L is selected, and we decrease the related counters. One counter reaches zero again ($B \wedge L \supset M$), so we add M to the agenda.

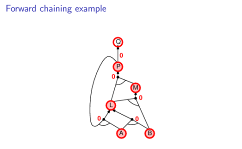
Forward chaining example



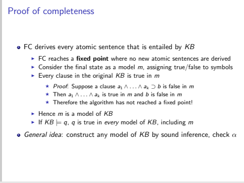
By selecting M, a counter again reaches zero, so P is added to the agenda.


Forward chaining example

Selecting P, Q is added to agenda, and there is an option to add L, but L has value, so we omit L.



We can only choose Q, but this was the query, so we can stop.

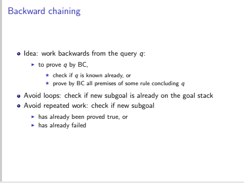


Proof of completeness

- FC derives every atomic sentence that is entailed by KB
  - FC reaches a **fixed point** where no new atomic sentences are derived
  - Consider the final state as a model m, assigning true/false to symbols
  - Every clause in the original KB is true in m
    - *Proof*: Suppose a clause $a_1 \land \ldots \land a_k \supset b$ is false in m
    - Then $a_1 \land \ldots \land a_k$ is true in m and b is false in m
    - Therefore the algorithm has not reached a fixed point!
  - Hence m is a model of KB
  - If $KB \models q$, q is true in *every* model of KB, including m
- *General idea*: construct any model of KB by sound inference, check α

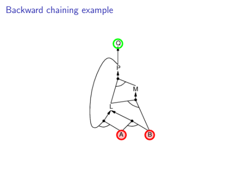
This method adds new facts to the KB – these facts are logical consequences of KB –, but we have finitely many variables, so this process must stop at some point, because there are no more new facts, i.e. we reach a fixpoint. Based on the extended KB we can construct an interpretation.

It is easy to prove, that all the statements in KB are true for this interpretation, so it is a model of it.

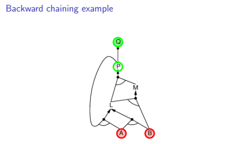

Backward chaining

- Idea: work backwards from the query q:
  - to prove q by BC,
    - check if q is known already, or
    - prove by BC all premises of some rule concluding q
- Avoid loops: check if new subgoal is already on the goal stack
- Avoid repeated work: check if new subgoal
  - has already been proved true, or
  - has already failed

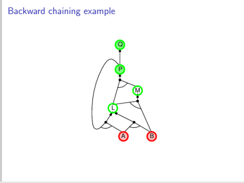
If KB has many consequences the forward chaining is a long process. Let us go in the opposite direction. Let the query q be a question that we want to prove. If q is known, we are ready. Otherwise we try to prove an implication with suffix q, so we need to prove (question) all the hypotheses. It is a search problem, but we need to do it in an effective way: we do not ask something twice.


Backward chaining example

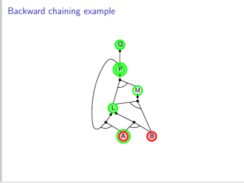
Our facts are A and B and we want to know whether Q is consequence of the KB. So let our question be Q.



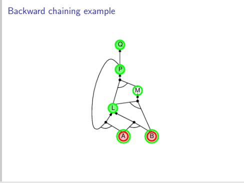
As Q is not a fact in the KB, does there exists an implication with suffix Q? The answer is yes, $P \supset Q$ is such an implication. So we have a subquestion P.



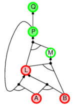
As P is not a fact, we have 2 subsubquestions L and M.



To answer L we need to ask P and A. P is already a subquestion, we do not ask it again. A is a fact, so the sub-sub question A is answered.

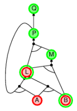


But L can be answered by A and B (an alternative answer). A has been answered and B is a fact, so it is also answered.

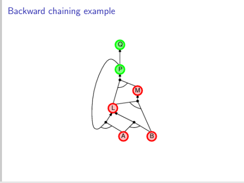


This means, that L is ready.

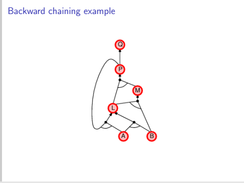
Backward chaining example



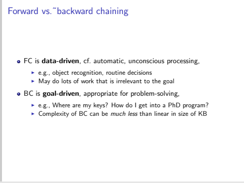
To answer M, we need to ask L and B, but both of them are ready,



so M is answered.



Similarly P and Q will be answered, and this concludes the search. From the description, BC seems longer, but when asking the relevant questions, it is usually faster than the FC.

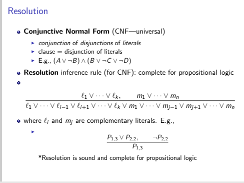


Forward vs.˜backward chaining

- FC is **data-driven**, cf. automatic, unconscious processing,
  - e.g., object recognition, routine decisions
  - May do lots of work that is irrelevant to the goal
- BC is **goal-driven**, appropriate for problem-solving,
  - e.g., Where are my keys? How do I get into a PhD program?
  - Complexity of BC can be *much less* than linear in size of KB

We use FC when we have no goals, just incoming data, and we need to manage these data in an autonomous way.

At BC we have some goal, and we search for answers. Typically the diagnostic (why this car does not start?), and classifying (so what is this animal?) problems are solved in this way.



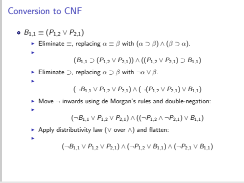
Resolution

- **Conjunctive Normal Form** (CNF—universal)
  - *conjunction* of *disjunctions* of *literals*
  - clause = disjunction of literals
  - E.g., $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$
- **Resolution** inference rule (for CNF): complete for propositional logic
- $$\frac{\ell_1 \vee \cdots \vee \ell_k, \qquad m_1 \vee \cdots \vee m_n}{\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n}$$
- where ℓ_i and m_j are complementary literals. E.g.,
  - $$\frac{P_{1,3} \vee P_{2,2}, \qquad \neg P_{2,2}}{P_{1,3}}$$

  *Resolution is sound and complete for propositional logic

We use Horn clauses in the following, but in clause form. For this we need to rewrite our formula into CNF form.
The resolution rule is a small modification of MP. For zero-order logic the resolution is a sound and complete method.

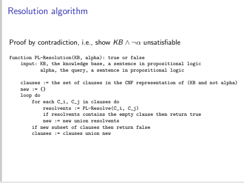
The formula below the line is called *resolvent*.



Conversion to CNF

- $B_{1,1} \equiv (P_{1,2} \lor P_{2,1})$
  - Eliminate $\equiv$, replacing $\alpha \equiv \beta$ with $(\alpha \supset \beta) \land (\beta \supset \alpha)$.
  - $$(B_{1,1} \supset (P_{1,2} \lor P_{2,1})) \land ((P_{1,2} \lor P_{2,1}) \supset B_{1,1})$$
  - Eliminate $\supset$, replacing $\alpha \supset \beta$ with $\neg\alpha \lor \beta$.
  - $$(\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land (\neg(P_{1,2} \lor P_{2,1}) \lor B_{1,1})$$
  - Move $\neg$ inwards using de Morgan's rules and double-negation:
  - $$(\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land ((\neg P_{1,2} \land \neg P_{2,1}) \lor B_{1,1})$$
  - Apply distributivity law ($\lor$ over $\land$) and flatten:
  - $$(\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land (\neg P_{1,2} \lor B_{1,1}) \land (\neg P_{2,1} \lor B_{1,1})$$

The slide Logical equivalence contains the necessary rules to rewrite any formula into CNF form. We need to rewrite formulae $R_1,\ldots,R_5$ into CNF form, see the example.



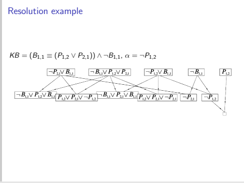
Resolution algorithm

Proof by contradiction, i.e., show $KB \wedge \neg\alpha$ unsatisfiable

```
function PL-Resolution(KB, alpha): true or false
    input: KB, the knowledge base, a sentence in propositional logic
           alpha, the query, a sentence in propositional logic

    clauses := the set of clauses in the CNF representation of (KB and not alpha)
    new := {}
    loop do
        for each C_i, C_j in clauses do
            resolvents := PL-Resolve(C_i, C_j)
            if resolvents contains the empty clause then return true
            new := new union resolvents
        if new subset of clauses then return false
        clauses := clauses union new
```

We take the negation of the formula α and the KB (in CNF) form, and construct all the resolvents. If we get a contradiction (an empty clause), we can stop, we have proved the entailment. Otherwise check that we have reached the fixpoint. If we have, we cannot prove the entailment, but we can construct a counterexample based on the extended KB. Otherwise, extend the KB with the set of resolvents and start over.

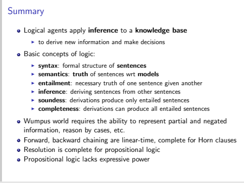


Resolution example

$KB = (B_{1,1} \equiv (P_{1,2} \vee P_{2,1})) \wedge \neg B_{1,1}$, $\alpha = \neg P_{1,2}$

$\neg P_{2,1} \vee B_{1,1}$
$\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}$
$\neg P_{1,2} \vee B_{1,1}$
$\neg B_{1,1}$
$P_{1,2}$
$\neg B_{1,1} \vee P_{1,2} \vee B_{1,1}$
$P_{1,2} \vee P_{2,1} \vee \neg P_{1,2}$
$\neg B_{1,1} \vee P_{2,1} \vee B_{1,1}$
$P_{1,2} \vee P_{2,1} \vee \neg P_{2,1}$
$\neg P_{2,1}$
$\neg P_{1,2}$

Here we have a restricted KB, and just one consequence. We need to rewrite the KB and the negation of formula alpha into CNF, and apply every possible resolution. We have reached the empty clause, so we are ready.



Summary

- Logical agents apply **inference** to a **knowledge base**
  - to derive new information and make decisions
- Basic concepts of logic:
  - **syntax**: formal structure of **sentences**
  - **semantics**: **truth** of sentences wrt **models**
  - **entailment**: necessary truth of one sentence given another
  - **inference**: deriving sentences from other sentences
  - **soundess**: derivations produce only entailed sentences
  - **completeness**: derivations can produce all entailed sentences
- Wumpus world requires the ability to represent partial and negated information, reason by cases, etc.
- Forward, backward chaining are linear-time, complete for Horn clauses
- Resolution is complete for propositional logic
- Propositional logic lacks expressive power

To manage knowledge bases we need to use logic, logical rules.
We refreshed the necessary concepts from introductory logic.
To be able to solve the simple game (Wumpus) we need to be able to manage information at a high-level.
We have seen effective methods: FC and BC that are used in the real life. (Clips & Prolog). Instead of MP we can use the resolution too.
For large problems zero-order logic is not enough, we need to use first-order logic (and the corresponding Horn formulae).