Knowledge-based agents

- Intelligent agents need **knowledge** about the world to choose good actions/decisions.
- Knowledge = **{sentences}** in a knowledge representation language (formal language).
- A sentence is an assertion about the world.
- A knowledge-based agent is composed of:
  1. Knowledge base: **domain-specific** content.
  2. Inference mechanism: **domain-independent** algorithms.

知识是基于特定领域的
算法是独立于领域的.

- 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
- **Declarative** approach to building an agent:
  - Add new sentences: **Tell** it what it needs to know
  - Query what is known: **Ask** itself what to do – answers should follow from the KB

```
function KB-AGENT(percept) returns an action
  static: KB, a knowledge base
          t, a counter, initially 0, indicating time
  TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
  action ← ASK(KB, MAKE-ACTION-QUERY(t))
  TELL(KB, MAKE-ACTION-SENTENCE(action, t))
  t ← t + 1
  return action
```

Logic

- **Knowledge base**: a set of sentences in a formal representation, **logic**
- **Logics**: are formal languages for representing knowledge to extract conclusions
  - **Syntax**: defines well-formed sentences in the language
  - **Semantic**: defines the truth or meaning of sentences in a world
- **Inference**: a procedure to derive a new sentence from other ones.
- **Logical entailment**: is a relationship between sentences. It means that a sentence **follows logically** from other sentences

$$KB \models \alpha$$

基于逻辑.推理,产生新的行为.

Propositional logic 命题逻辑

- Propositional logic (PL) is the simplest logic.
- **Syntax of PL:** defines the allowable sentences or propositions.
- **Definition (Proposition):** A proposition is a declarative statement that's either True or False.
- **Atomic proposition:** single proposition symbol. Each symbol is a proposition. Notation: upper case letters and may contain subscripts.
- **Compound proposition:** constructed from atomic propositions using parentheses and **logical connectives**.

Atomic proposition

Examples of atomic propositions:

- 2+2=4 is a true proposition
- $W_{1,3}$ is a proposition. It is true if there is a Wumpus in [1,3]
- "How are you?" or "Hello!" are not propositions. In general, statement that are questions, commands, or opinions are not propositions.

Compound proposition

Examples of compound/complex propositions:

Let p, p_1, and p_2 be propositions

- **Negation** $\neg p$ is also a proposition. We call a **literal** either an atomic proposition or its negation. E.g., $W_{1,3}$ is a positive literal, and $\neg W_{1,3}$ is a negative literal.
- **Conjunction** $p_1 \wedge p_2$. E.g., $W_{1,3} \wedge P_{3,1}$
- **Disjunction** $p_1 \vee p_2$. E.g., $W_{1,3} \vee P_{3,1}$
- **Implication** $p_1 \rightarrow p_2$. E.g., $W_{1,3} \wedge P_{3,1} \rightarrow \neg W_{2,2}$
- **If and only if** $p_1 \leftrightarrow p_2$. E.g., $W_{1,3} \leftrightarrow \neg W_{2,2}$

Truth tables

- The semantics define the rules to determine the truth of a sentence.
- Semantics can be specified by truth tables.
- Boolean values domain: T, F
- n-tuple: $(x_1, x_2, ..., x_n)$
- Operator on n-tuples: $g(x_1 = v_1, x_2 = v_2, ..., x_n = v_n)$
- Definition: A truth table defines an operator g on n-tuples by specifying a Boolean value for each tuple
- Number of rows in a truth table? $R = 2^n$

Buiding propositions

Negation:

p	$\neg p$
T	F
F	T

Conjunction:

p	q	$p \wedge q$
T	T	T
T	F	F
F	T	F
F	F	F

Disjunction:

p	q	$p \vee q$
T	T	T
T	F	T
F	T	T
F	F	F

p	q	r	¬r	p ∨ q	p ∨ q → ¬r
T	T	T	F	T	F
T	T	F	T	T	T
T	F	T	F	T	F
T	F	F	T	T	T
F	T	T	F	T	F
F	T	F	T	T	T
F	F	T	F	F	T
F	F	F	T	F	T

Exclusive or:

p	q	$p \oplus q$
T	T	F
T	F	T
F	T	T
F	F	F

Implication:

p	q	$p \rightarrow q$
T	T	T
T	F	F
F	T	T
F	F	T

Biconditional or If and only if (IFF):

p	q	$p \leftrightarrow q$
T	T	T
T	F	F
F	T	F
F	F	T

Precedence of operators

- Just like arithmetic operators, there is an operator precedence when evaluating logical operators as follows:
  1. Expressions in parentheses are processed (inside to outside)
  2. Negation
  3. AND
  4. OR
  5. Implication
  6. Biconditional
  7. Left to right
- **Use parentheses whenever you have any doubt!**

Logical equivalence

- Two propositions p and q are logically equivalent if and only if the columns in the truth table giving their truth values agree.
- We write this as $p \equiv q$.

p	q	$\neg p$	$\neg p \vee q$	$p \rightarrow q$
T	T	F	T	T
T	F	F	F	F
F	T	T	T	T
F	F	T	T	T

Properties of operators

- Commutativity:
  $p \wedge q \equiv q \wedge p$
  $p \vee q \equiv q \vee p$
- Associativity:
  $(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$
  $(p \vee q) \vee r \equiv q \vee (p \vee r)$
- Identity element:
  $p \wedge \mathit{True} \equiv p$
  $p \vee \mathit{True} \equiv \mathit{True}$
- $\neg(\neg p) \equiv p$
- $p \wedge p \equiv p$ and $p \vee p \equiv p$
- Distributivity:
  $p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$
  $p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$
- $p \wedge (\neg p) \equiv \mathit{False}$ and $p \vee (\neg p) \equiv \mathit{True}$
- DeMorgan's laws:
  $\neg(p \wedge q) \equiv (\neg p) \vee (\neg q)$
  $\neg(p \vee q) \equiv (\neg p) \wedge (\neg q)$

Tautology and contradiction

- **Tautology** is a proposition which is always true
- **Contradiction** is a proposition which is always false
- **Contingency** is a proposition which is neither a tautology or a contradiction

恒真
恒假
不定式

P	$\neg p$	$p \vee \neg p$	$p \wedge \neg p$
T	F	T	F
F	T	T	F

Contrapositive, inverse, etc.

- Given an **implication** $p \rightarrow q$
- The **converse** is: $q \rightarrow p$
- The **contrapositive** is: $\neg q \rightarrow \neg p$
- The **inverse** is: $\neg p \rightarrow \neg q$

Example: Hot is a sufficient condition for my going to the beach.

- The **implication** is:
- The **converse** is:
- The **contrapositive** is:
- The **inverse** is:

Inference (Modus Ponens)

$$\frac{p \quad p \to q}{q}$$

$$\frac{warm \quad warm \to sunny}{sunny}$$

Horn clauses: a proposition of the form:

$$p_1 \wedge \ldots \wedge p_n \to q$$

Modus Ponens deals with **Horn clauses**:

$$\frac{p_1, \ldots, p_n \quad (p_1 \wedge \cdots \wedge p_n) \to q}{q}$$

Inference (Modus Tollens)

$$\frac{\neg q \quad p \to q}{\neg p}$$

$$\frac{\neg beach \quad hot \to beach}{\neg hot}$$

Common Rules

- Addition: $\frac{p}{p \vee q}$
- Simplification: $\frac{p \wedge q}{q}$
- Disjunctive-syllogism: $\frac{p \vee q \quad \neg p}{q}$
- Hypothetical-syllogism: $\frac{p \to q \quad q \to r}{p \to r}$

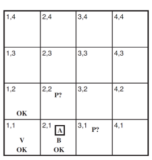
Truth Tables for connectives

Summary:

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
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*

Example

Wumpus world KB



- Let's build the KB for the reduced Wumpus world.
- 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 $[i, j]$

  R_1: $\neg P_{1,1}$
- "A square is breezy if and only if there is an adjacent pit"

  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: $B_{2,1}$
- Questions: Based on this KB, is $KB \models P_{1,2}$? Is $KB \models P_{2,2}$?

35

Entailment and Inference

- **Semantics**: Determine entailment by **Model Checking**, that is enumerate all models and show that the sentence must hold in all models.

$$KB \models \alpha$$

- **Syntax**: Determine entailment by **Theorem Proving**, that is apply rules of inference to KB to build a proof of α *without enumerating and checking all models.*

$$KB \vdash \alpha$$

- But how are entailment and inference related?

Soundness & Completeness

- We want an inference algorithm that is:
  1. **Sound**: does not infer false formulas, that is, derives only entailed sentences.
     $$\{\alpha | KB \vdash \alpha\} \subseteq \{\alpha | KB \models \alpha\}$$
  2. **Complete**: derives ALL entailed sentences.
     $$\{\alpha | KB \vdash \alpha\} \supseteq \{\alpha | KB \models \alpha\}$$

Validity & satisfiability

- A sentence is **valid** (aka tautology) if it is true in all models, e.g., *True*, $p \vee \neg p$, $p \Rightarrow p$, $(p \wedge (p \Rightarrow q)) \Rightarrow q$
- Validity is connected to inference via the **Deduction Theorem**: $KB \models \alpha$ IFF $(KB \Rightarrow \alpha)$ is valid
- A sentence is **satisfiable** if it is true in some model, e.g., $p \vee p$, r
- A sentence is **unsatisfiable** if it is true in no models, e.g., $p \wedge \neg p$
- Satisfiability is connected to inference via the following: $KB \models \alpha$ IFF $(KB \wedge \neg \alpha)$ is unsatisfiable, i.e., prove α by contradiction

Determining entailment

- Given a Knowledge Base (KB) (set of sentences in PL), given a query α, output whether KB entails α, noted: $KB \models \alpha$
- We will see two ways of doing proofs in PL:
  - **Model checking** enumerate the models (truth table enumeration, exponential).
  - **Application of inference rules** (proof checking/theorem proving): Syntactic derivations with rules like Modus Ponens (Backward chaining and forward chaining). A proof is a sequence of inference rule applications.

Model Checking

- Truth Table for inference
- Model: assignment T/F to every propositional symbol.
- Check that is true in every model in which KB is true.

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	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	true
false	true	false	false	false	true	false	true	true	true	true	true	true
false	true	false	false	false	true	true	true	true	true	true	true	true
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

Inference as a search problem

- **Initial state:** The initial KB
- **Actions:** all inference rules applied to all sentences that match the top of the inference rule
- **Results:** add the sentence in the bottom half of the inference rule
- **Goal:** a state containing the sentence we are trying to prove.

Theorem Proving

- Search for proofs is a more efficient way than enumerating models (We can ignore irrelevant information)
- Truth tables have an exponential number of models.
- The idea of inference is to repeat applying *inference rules* to the KB.
- Inference can be applied whenever suitable premises are found in the KB.
- Inference is sound. How about completeness?
- Two ways to ensure completeness:
  - **Proof by resolution**: use powerful inference rules (resolution rule)
  - **Forward or Backward chaining**: use of modus ponens on a restricted form of propositions (Horn clauses)
- Resolution: ONE single inference rule
- Invented by Robinson, 1965
- Resolution + Search = complete inference algorithm.

Prove by Resolution

- **Unit resolution:**

$$\frac{\ell_1 \vee \cdots \vee \ell_k \qquad m}{\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k}$$

where ℓ_i and m are complementary literals.

- Example:

$$\frac{P_{1,3} \vee P_{2,2} \qquad \neg P_{2,2}}{P_{1,3}}$$

- We call a **clause** a disjunction of literals.
- Unit resolution: Clause + Literal = New clause.

- **Resolution inference rule (for CNF):**

$$\frac{\ell_1 \vee \cdots \vee \ell_k \qquad m_1 \vee \cdots \vee m_n}{\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n}$$

where ℓ_i and m_j are complementary literals.

- Resolution **applies only to clauses**
- **Fact**: Every sentence in PL is logically equivalent to a conjunction of clauses.
- Conjunctive Normal Form (CNF): Conjunction of disjunction of literals: example: $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$
- Resolution inference rule (for CNF): sound and complete for propositional logic

Conversion to CNF

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

1. Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$.
   $(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$
2. Eliminate $\Rightarrow$, replacing $\alpha \Rightarrow \beta$ with $\neg\alpha \vee \beta$.
   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$
3. Move $\neg$ inwards using DeMorgan's rules and double-negation:
   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$
4. Apply distributivity law ($\vee$ over $\wedge$) and flatten:
   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$

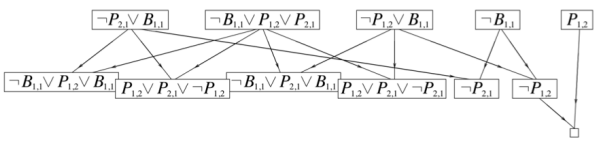
Resolution algorithm

```
function PL-RESOLUTION(KB, α) returns true or false
  inputs: KB, the knowledge base, a sentence in propositional logic
          α, the query, a sentence in propositional logic
  clauses ← the set of clauses in the CNF representation of KB ∧ ¬α
  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 ∪ resolvents
    if new ⊆ clauses then return false
    clauses ← clauses ∪ new
```

Resolution example

$KB = R_4 \wedge R_2 = (B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})) \wedge \neg B_{1,1}$

$\alpha = \neg P_{1,2}$



Forward / backward chaining

- KB = conjunction of Horn clauses
- Horn clauses: logic proposition of the form: $p_1 \wedge \ldots \wedge p_n \rightarrow q$
- Modus Ponens (for Horn Form): complete for Horn KBs

$$\frac{p_1, \ldots, p_n \quad (p_1 \wedge \cdots \wedge p_n) \rightarrow q}{q}$$

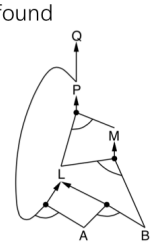
- Can be used with **forward chaining** or **backward chaining**.
- These algorithms are very natural and run in linear time

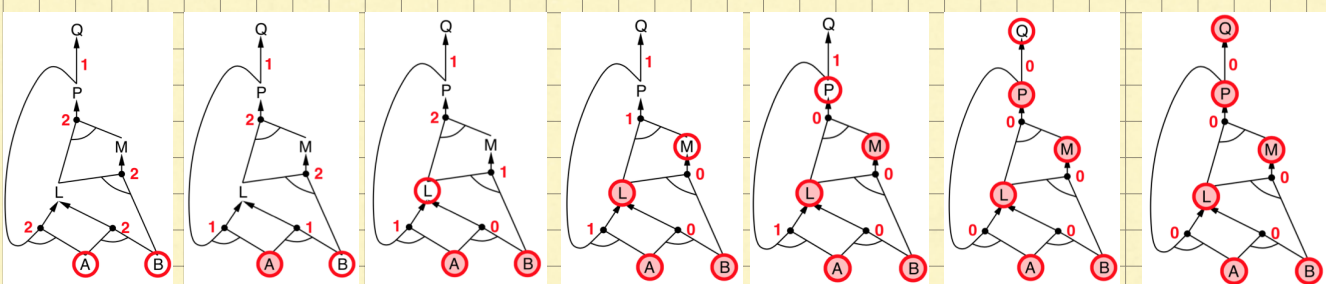
Forward chaining

Idea:

Fire any rule whose premises are satisfied in the KB, add its conclusion to the KB, until query is found

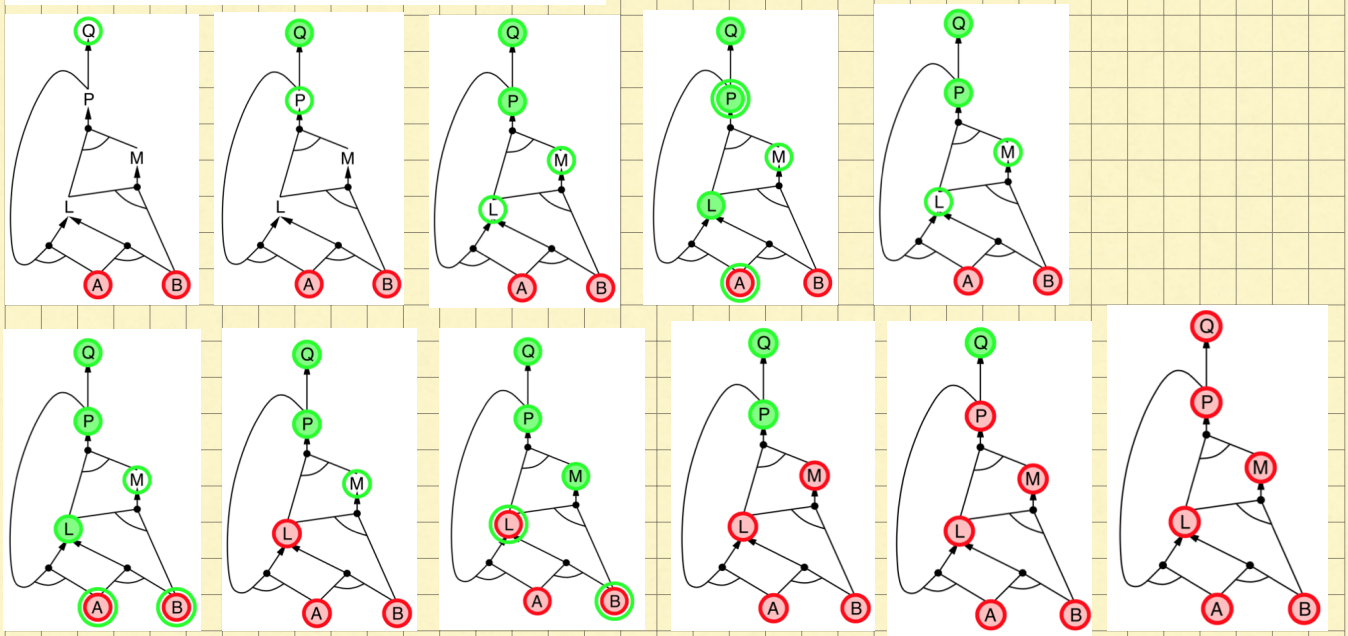
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B





Backward chaining

Idea: Works backwards from the query q
- to prove q by Backward Chaining:
  - Check if q is known already, or
  - Prove by Backward Chaining 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



Forward vs. Backward

- Forward chaining:
  - Data-driven, automatic, unconscious processing,
  - May do lots of work that is irrelevant to the goal
- Backward chaining:
  - Goal-driven, appropriate for problem-solving,
  - Complexity of BC can be much less than linear in size of KB

Propositional Logic

- Propositional Logic (PL) is a formal language to describe the world around us.
- Logic can be used by an agent to model the world.
- Sentences in PL have a fixed **syntax**.
- With symbols and connectives we can form logical sentences:
  - Symbols or terms that can be either True or False or unknown.
  - Logical connectives

  Example: $hot \wedge sunny \Rightarrow beach \vee pool$
- Syntax and **Semantics** represent two important and distinct aspects in PL.
- Semantic: configurations/instantiations of the world.

- Modus Ponens inference rule:

$$\frac{p_1, \ldots, p_n \quad (p_1 \wedge \cdots \wedge p_n) \rightarrow q}{q}$$

- Example:

$$\frac{warm \quad warm \rightarrow sunny}{sunny}$$

- Modus Ponens deals with Horn clauses
- Horn clauses: logic proposition of the form: $p_1 \wedge \ldots \wedge p_n \rightarrow q$
- **Inference**: we want an inference algorithm that is:
  1. sound (does not infer false formulas), and
  2. ideally, complete too (derives all true formulas).
- **Inference in PL with horn clauses is sound and complete.**

- Limits of PL?
  1. PL is not expressive enough to describe all the world around us. It can't express information about different object and the relation between objects.
  2. PL is not compact. It can't express a fact for a set of objects without enumerating all of them which is sometimes impossible.
- Example: We have a vacuum cleaner (Roomba) to clean a 10×10 squares in the classroom. Use PL to express information about the squares.

- The proposition $square_1\_is\_clean$ expresses information about square1 being clean. How can one write this in a less heavy way?
- How can we express that all squares in the room are clean?

  $square_1\_is\_clean \wedge square_2\_is\_clean \wedge \ldots \wedge square_{100}\_is\_clean$
- How can we express that some squares in the room are clean?

  $square_1\_is\_clean \vee square_2\_is\_clean \vee \ldots \vee square_{100}\_is\_clean$
- How can we express that some squares have chairs on them?

  $square_1\_has\_chair \vee square_2\_has\_chair \vee \ldots \vee square_{100}\_has\_chair$

First Order Logic

- Alternative to PL: Another more powerful language, **First Order Logic (FOL).**
- Syntax of FOL:
  - Terms are either:
    - Constant symbols (e.g., A, 10, Shenzhen),
    - Variables (e.g., *x, y*)
    - Functions of terms, e.g., sqrt(x), sum(1,2).
  - Atomic formulas: predicates applied to terms, e.g., brother(x,y), above(A,B)
  - Connectives: ∧, ∨, ⇒, ⇔, ¬
  - Equality: ≡
  - Quantifiers: ∀, ∃
  - Connectives, equality, quantifiers can be applied to atomic formulas to create sentences in FOL.

78

All squares are clean:

$$\forall x \ \ Square(x) \Rightarrow Clean(x)$$

There exists some dirty squares:

$$\exists x \ \ Square(x) \wedge \neg Clean(x)$$

Note:

- $\forall x \ \ P(x)$ is like $P(A) \wedge P(B) \wedge \ldots$
- $\exists x \ \ P(x)$ is like $P(A) \vee P(B) \vee \ldots$
- $\neg \forall x \ \ P(x)$ is like $\exists x \ \neg P(x)$
- $\forall x \ \exists y \ \ likes(x, y)$ is NOT like $\exists y \ \forall x \ \ likes(x, y)$

- All birds fly:

  $$\forall x \ \ bird(x) \Rightarrow Fly(x)$$
- All birds except penguins fly:

  $$\forall x \ \ bird(x) \wedge \neg penguin(x) \Rightarrow Fly(x)$$
- Every kid likes candy:

  $$\forall x \ \ Kid(x) \Rightarrow Likes(x, candy)$$
- Some kids like candy:

  $$\exists x \ \ Kid(x) \wedge Likes(x, candy)$$
- Brothers are sibling:

  $$\forall x, y \ \ Brothers(x, y) \Rightarrow Sibling(x, y)$$
- One's mother is one's female parent:

  $$\forall x, y \ \ Mother(x, y) \Leftrightarrow Female(x) \wedge Parent(x, y)$$

- **Inference:** While a bit more complicated than PL, there are procedures to do inference with a knowledge base of FOL formulas (Further optional reading: book chapter 8, 9).
- **Natural language:** The expressiveness of FOL suggests that it is possible to automate the conversion between natural language and logical expressions. This is very valuable for different applications, such as personal assistants (Siri), question/answering systems, and communicating with computers in general.

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
  - **Soundness**: derivations produce only entailed sentences
  - **Completeness**: derivations can produce all entailed sentences
- Forward, backward chaining are linear in time, complete for Horn clauses. Resolution is complete for propositional logic.

- Building logical agents was a main research trend in AI before the mid-nineties
- Logic is used in AI to represent the environment of the agent and reason about that environment
- Pros and cons of logical agents:
  - Do not handle uncertainty
  - Rule-based and do not use data (Machine Learning)
  - It is hard to model every aspect of the world
  + Intelligibility of models: models are encoded explicitly

Backtracking for SAT

- Satisfiability is connected to inference via the following: $KB \models \alpha$ IFF $(KB \wedge \neg\alpha)$ is **unsatisfiable**, i.e., prove α by contradiction
- The DPLL algorithm for checking **satisfiability** (SAT) of a sentence in propositional logic.
- The DPLL algorithm is similar to Backtracking for CSP, but using various problem dependent information/heuristics, such as **Early Termination**, **Pure symbol heuristic** and **Unit clause heuristic**.

DPLL

function DPLL-SATISFIABLE?(s) **returns** *true* or *false*
  inputs: s, a sentence in propositional logic

  clauses ← the set of clauses in the CNF representation of s
  symbols ← a list of the proposition symbols in s
  return DPLL(*clauses*, *symbols*, { })

---

function DPLL(*clauses*, *symbols*, *model*) **returns** *true* or *false*

  if every clause in *clauses* is true in *model* **then return** *true*
  if some clause in *clauses* is false in *model* **then return** *false*
  P, *value* ← FIND-PURE-SYMBOL(*symbols*, *clauses*, *model*)
  if P is non-null **then return** DPLL(*clauses*, *symbols* – P, *model* ∪ {P=*value*})
  P, *value* ← FIND-UNIT-CLAUSE(*clauses*, *model*)
  if P is non-null **then return** DPLL(*clauses*, *symbols* – P, *model* ∪ {P=*value*})
  P ← FIRST(*symbols*); *rest* ← REST(*symbols*)
  return DPLL(*clauses*, *rest*, *model* ∪ {P=*true*}) **or**
        DPLL(*clauses*, *rest*, *model* ∪ {P=*false*}))