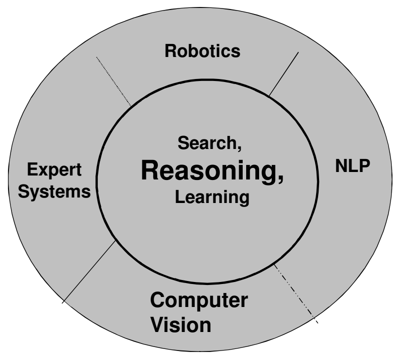
Reasoning

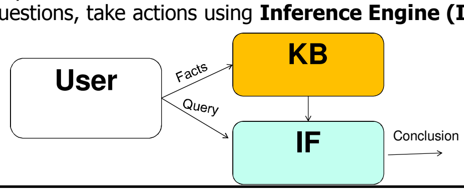


Inference

- Logical Inference (deduction) derives new sentences in the language from existing ones,.
  - Socrates is a man.
  - <u>All men are mortal.</u>
  - Socrates is mortal.
- Proper inference should only derive sound conclusions

Representation and Reasoning

- In order to determine appropriate actions to take, an intelligent system needs to represent information about the world and draw conclusions based on general world knowledge and specific facts.
- Knowledge is represented by sentences in some language stored in a **knowledge base (KB).**
- A system draws conclusions from the KB to answer questions, take actions using **Inference Engine (IF)**.



Propositional calculus & First-order logic

- Propositional logic assumes world contains facts.
- First-order logic (like natural language) assumes the world contains
- Objects: people, houses, numbers, …
- Relations: red, round, prime,…
- Functions: fatherof, friend, in,…
- Propositional calculus
  $A \wedge B \Rightarrow C$
- First-order predicate calculus
  $(\forall x)(\exists y)$ Mother(y,x)

Knowledge Representation

- 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
    syntax
  - $x+2 \geq y$ is true in a world where $x = 7, y = 1$
  - $x+2 \geq y$ is false in a world where $x = 0, y = 6$

  } semantics

Syntax of PC

- Connectives: $\neg, \wedge, \vee, \Rightarrow$
- Propositional symbols, e.g., P, Q, R, …
  - *True, False*
  - **Syntax of PC**
- sentence → atomic sentence | complex sentence
- atomic sentence → Propositional symbol, *True, False*
- Complex sentence → $\neg$sentence
  | (sentence $\wedge$ sentence)
  | (sentence $\vee$ sentence)
  | (sentence $\Rightarrow$ sentence)
  - **Rules of Inference:**
  - Ex: Modus ponens

Semantics of PC

A	B	$\neg$ A	A $\wedge$ B	A $\vee$ B	A $\Rightarrow$ B
True	*True*	*False*	*True*	*True*	*True*
True	*False*	*False*	*False*	*True*	*False*
False	*False*	*True*	*False*	*False*	*True*
False	*True*	*True*	*False*	*True*	*True*

Rules of Inference

- **Modus Ponens:** $\{\alpha \Rightarrow \beta, \alpha\} \vdash \beta$
- **And Elimination:** $\{\alpha \wedge \beta\} \vdash \alpha$; $\{\alpha \wedge \beta\} \vdash \beta$
- **Double negation Elimination:** $\{\neg\neg\alpha\} \vdash \alpha$
- **Implication Elimination** $\{\alpha \Rightarrow \beta\} \vdash \neg\alpha \vee \beta$
- **Unit resolution:** $\{\alpha \vee \beta, \neg\beta\} \vdash \alpha$
- **Resolution:** $\{\alpha \vee \beta, \neg\beta \vee \gamma\} \vdash \alpha \vee \gamma$

Validity and Inference

- An Interpretation is an assignment of a truth value (True or False) to each atomic proposition
- A sentence that is true under all interpretation is valid (i.e. tautology)
- Validity can be checked by the truth table
- Inference can be done by checking the validity of each sentence. (may be applying truth table)
- An alternative to checking all rows of a truth table, one can use rules of inference to draw conclusions.

Models and Entailment

- A model is any interpretation in which a statement is true.
- A sentence A **entails** B (A $\models$ B) if every model of A is Also a model of B. i.e. if A is true then B must be true
- A statement B is entailed from some KB if there is a logical inference to deduce B

  KB $\models$ B if KB→B

Rules of Inference

- A sequence of inference rule applications that leads to a desired conclusion is called a **logical proof.**
- A |- B , denotes that B can be derived by some inference procedure from the set of sentences A.
- Inference rules can be verified by the truth-table
- Then used to construct **sound proofs.**
- Finding a proof is simply a **search problem** with the inference rules as operators and the conclusion as the goal

Satisfiability

- A sentence is **satisfiable** if it is true under some interpretation (i.e. it has a model), otherwise the sentence is **unsatisfiable.**
- A sentence is **valid** if and only if its negation is unsatisfiable.
- •Therefore, algorithms for either validity or satisfiability
- checking are useful for logical inference.
- If there are *n propositional symbols in a sentence, then we* must check 2^n *rows for validity*
- **Satisfiability is** NP-complete, i.e. there is no polynomial-time algorithm to solve.
- Yet, many problems can be solved very quickly.

Pros and cons of propositional logic

- ✓ Propositional logic is declarative: pieces of syntax correspond to facts
- ✓ Propositional logic is compositional: meaning of A ^ B is derived from meaning of A and B
- ✓ Meaning in propositional logic is context-independent
- (unlike natural language, where meaning depends on context)
- ➢ Propositional logic has very limited expressive power
- (unlike natural language)

FOL Syntax

- **Variable symbols**
  - E.g., x, y, John
- **Connectives**: ¬, ∧, ∨, ⇒
  - **Quantifiers**
  - Universal ∀x
  - Existential ∃x

Propositional logic is a weak language

- Hard to identify "individuals." Ex. Mary, 3
- Can't directly talk about properties of individuals or relations between individuals. Ex. "Bill is tall"
- Generalizations, patterns, regularities can't easily be represented. Ex. all triangles have 3 sides
- **First-Order Logic** (abbreviated FOL or FOPC) is expressive enough to concisely represent this kind of situation.
  - FOL adds relations, variables, and quantifiers, e.g.,
    - "Every elephant is gray": ∀ x (elephant(x) → gray(x))
    - "There is a white elephant": ∃ x (elephant(x) ^ white(x))

Sentence → *Atomicsentence*
| (*Sentence Connective Sentence*)
| *Quantifier Variable,. . . Sentence*
| *Sentence*
AtomicSentence → *Predicate(Term, . . .)*
| (*Term* = *Term*
Term -→ *Function(Term, . . .)*
I *Constant*
| *Variable*
Connective → ¬, ∧, ∨, ⇒
Quantifier → ∀, ∃
Constant → *A* (XI (*John* 1 . . .
Variable → *a* | *x* | s | . . .
Predicate → *Before…*
Function → *Mother* | …

First-order logic

- First-order logic (FOL) models the world in terms of
  - Objects, which are things with individual identities
  - Properties of objects that distinguish them from other objects
  - Relations that hold among sets of objects
  - Functions, which are a subset of relations where there is only one "value" for any given "input"

Ex:Objects: Students, lectures, companies, cars ...
  - Relations: Brother-of, bigger-than, outside, part-of, has-color, occurs-after, owns, visits, precedes, ...
  - Properties: blue, oval, even, large, ...
  - Functions: father-of, best-friend, second-half, one-more-than ...

Nested Quantifiers

- Combinations of universal and existential quantification are possible:

$$\forall x \forall y\ Father(x,y) \equiv \forall y \forall x\ Father(x,y)$$
$$\exists x \exists y\ Father(x,y) \equiv \exists y \exists x\ Father(x,y)$$
$$\forall x \exists y\ Father(x,y) \neq \exists y \forall x\ Father(x,y)$$
$$\exists x \forall y\ Father(x,y) \neq \forall y \exists x\ Father(x,y)$$
$$x, y \in \{All\ \ people\}$$

Logical equivalence in PC

- Two sentences are logically equivalent iff true in same models: $\alpha \equiv \beta$ iff $\alpha \models \beta$ and $\beta \models \alpha$
- EXamples:

$$
\begin{array}{rcll}
(\alpha \wedge \beta) & \equiv & (\beta \wedge \alpha) & \text{commutativity of } \wedge \\
(\alpha \vee \beta) & \equiv & (\beta \vee \alpha) & \text{commutativity of } \vee \\
((\alpha \wedge \beta) \wedge \gamma) & \equiv & (\alpha \wedge (\beta \wedge \gamma)) & \text{associativity of } \wedge \\
((\alpha \vee \beta) \vee \gamma) & \equiv & (\alpha \vee (\beta \vee \gamma)) & \text{associativity of } \vee \\
\neg(\neg\alpha) & \equiv & \alpha & \text{double-negation elimination} \\
(\alpha \Rightarrow \beta) & \equiv & (\neg\beta \Rightarrow \neg\alpha) & \text{contraposition} \\
(\alpha \Rightarrow \beta) & \equiv & (\neg\alpha \vee \beta) & \text{implication elimination} \\
(\alpha \Leftrightarrow \beta) & \equiv & ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) & \text{biconditional elimination} \\
\neg(\alpha \wedge \beta) & \equiv & (\neg\alpha \vee \neg\beta) & \text{de Morgan} \\
\neg(\alpha \vee \beta) & \equiv & (\neg\alpha \wedge \neg\beta) & \text{de Morgan} \\
(\alpha \wedge (\beta \vee \gamma)) & \equiv & ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) & \text{distributivity of } \wedge \text{ over } \vee \\
(\alpha \vee (\beta \wedge \gamma)) & \equiv & ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) & \text{distributivity of } \vee \text{ over } \wedge
\end{array}
$$

Properties of quantifiers

∀x ∀y is the same as ∀y ∀x
∃x ∃y is the same as ∃y ∃x

∃x ∀y is not the same as ∀y ∃x
∃x ∀y Loves(x,y)
- "There is a person who loves everyone in the world"

∀y ∃x Loves(x,y)
- "Everyone in the world is loved by at least one person"

- Quantifier duality: each can be expressed using the other

Exp.	Negation
∀x Likes(x,IceCream)	∃x ¬Likes(x,IceCream)
∃x Likes(x,Broccoli)	∀x ¬Likes(x,Broccoli)

A common mistake to avoid

- Typically, ⇒ is the main connective with ∀
- Common mistake: using ∧ as the main connective with ∀:
- Ex:

∀x At(x,CU) ∧ Smart(x)
means "Everyone is at CU and everyone is smart"

Yet to say Everyone at CU is smart
∀x At(x,CU) ⇒ Smart(x)

Equality

Equality:

term$_1$ = *term*$_2$ is true under a given interpretation if and only if *term*$_1$ and *term*$_2$ refer to the same object

FOPC can include equality as a primitive predicate or require it to be as identity relation

Equal(x,y) or x=y

Examples:

to say "that Mary is taking two courses", you need to insure that x,y are different

∃x ∃y (takes(Mary,x) ^ takes (Mary,y) ^ ~ (x=y))

To say "Everyone has exactly one father"

∀x ∃y father(y,x) ^ ∀z father(z,x) ➔ y=z

Another common mistake to avoid

- Typically, ∧ is the main connective with ∃
- Common mistake: using ⇒ as the main connective with ∃:

∃*x* At(x,CU) ⇒ Smart(x)

is true if there is anyone who is smart not at CU.

Yet to say: there exists someone in CU that is smart
∃*x* At(x,CU) ∧ Smart(x)

Higher Order Logic

- FOPC is called first order because it allows quantifiers to rang only over objects (terms).

  ∀x, ∀y [*x=y or x>y or y>x*]

- **Second-Order Logic** allows quantifiers to range over predicates and functions as well

  ∀f, ∀g [*f=g* <=> (∀x f(x)=g(x))]

- **Third-Order Logic** allows quantifiers to range over predicates of predicates,.. etc

Types of Logic

- Propositional Logic : Facts {0,1}

  Also called Propositional Calculus (denoted as PC)
- First Order predicate Calculus : Facts, Objects, Relations {0,1}

  Also called First Order Logic (denoted FOL)
- Temporal Order Logic : Facts, Objects, Relations, Time {0,1}
- Probability Theory: Fast --- Degree of belief [0,1]
- Fuzzy Logic: Degree of Truth. [0,1]

Translating English to FOL

- Every gardener likes the sun.
  `(∀x) gardener(x) => likes(x,Sun)`
- You can fool some of the people all of the time.

Examples of FOPC

- Brothers are siblings

  ∀x, ∀y *Brother(x,y)* => *Sibling(x,y)*
- One's mother is one's female parent

  ∀m, ∀c *Mother(c)* = m ⇔ *(Female(m)* ∧ *Parent(m,c))*
- "Sibling" is symmetric

  ∀x, ∀y *Sibling(x,y)* ⇔ *Sibling(y,x)*

Translating English to FOL

- Every gardener likes the sun.
  `(∀x) gardener(x) => likes(x,Sun)`
- You can fool some of the people all of the time.
  `(∃x) person(x) ^ ((∀t) time(t)) => can-fool(x,t))`
- You can fool all of the people some of the time.

Translating English to FOL

- Every gardener likes the sun.

Translating English to FOL

- Every gardener likes the sun.
  `(∀x) gardener(x) => likes(x,Sun)`
- You can fool some of the people all of the time.
  `(∃x) person(x) ^ ((∀t) time(t)) => can-fool(x,t))`
- You can fool all of the people some of the time.
  `(∀x) person(x) => ((∃t) time(t) ^ can-fool(x,t))`
- All purple mushrooms are poisonous.

Translating English to FOL

- Every gardener likes the sun.
  `(∀x) gardener(x) => likes(x,Sun)`
- You can fool some of the people all of the time.
  `(∃x) person(x) ^ ((∀t) time(t)) => can-fool(x,t))`
- You can fool all of the people some of the time.
  `(∀x) person(x) => ((∃t) time(t) ^ can-fool(x,t))`
- All purple mushrooms are poisonous.
  `(∀ x) (mushroom(x) ^ purple(x)) => poisonous(x)`
- No purple mushroom is poisonous.

Translating English to FOL

- Every gardener likes the sun.
  `(∀x) gardener(x) => likes(x,Sun)`
- You can fool some of the people all of the time.
  `(∃x) person(x) ^ ((∀t) time(t)) => can-fool(x,t))`
- You can fool all of the people some of the time.
  `(∀x) person(x) => ((∃t) time(t) ^ can-fool(x,t))`
- All purple mushrooms are poisonous.
  `(∀ x) (mushroom(x) ^ purple(x)) => poisonous(x)`
- No purple mushroom is poisonous.
  `~(∃x) purple(x) ^ mushroom(x) ^ poisonous(x)`
  or, equivalently,
  `(∀x) (mushroom(x) ^ purple(x)) => ~poisonous(x)`
- There are exactly two purple mushrooms.

Translating English to FOL

- Every gardener likes the sun.
  `(∀x) gardener(x) => likes(x,Sun)`
- You can fool some of the people all of the time.
  `(∃x) person(x) ^ ((∀t) time(t)) => can-fool(x,t))`
- You can fool all of the people some of the time.
  `(∀x) person(x) => ((∃t) time(t) ^ can-fool(x,t))`
- All purple mushrooms are poisonous.
  `(∀ x) (mushroom(x) ^ purple(x)) => poisonous(x)`
- No purple mushroom is poisonous.
  `~(∃x) purple(x) ^ mushroom(x) ^ poisonous(x)`
  or, equivalently,
  `(∀x) (mushroom(x) ^ purple(x)) => ~poisonous(x)`
- There are exactly two purple mushrooms.
  `(∃x)(∃y) mushroom(x) ^ purple(x) ^ mushroom(y) ^ purple(y) ^ ~(x=y) ^ (∀z) (mushroom(z) ^ purple(z)) => ((x=z) v (y=z))`

Exercises

- Not all students take both History & Biology
- Only one student failed History
- Only one student failed both History & Biology
- The best score in History is better than the best score in Biology
- No person likes a professor unless the professor is smart
- Politicians can fool some of the people all the time, and they can fool all the people some of the time, but they can not fool all the people all the time.