## 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)

# 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 ...

# A common mistake to avoid

- Typically, $\Rightarrow$ is the main connective with $\forall$
- Common mistake: using $\wedge$ as the main connective with $\forall$:
- Ex:

$\forall x\ At(x,CU) \wedge Smart(x)$
means “Everyone is at CU and everyone is smart”

Yet to say Everyone at CU is smart
$\forall x\ At(x,CU) \Rightarrow Smart(x)$

## Another common mistake to avoid

- Typically, $\wedge$ is the main connective with $\exists$
- Common mistake: using $\Rightarrow$ as the main connective with $\exists$:

  $$\exists \boldsymbol{x}\, At(x,CU) \Rightarrow Smart(x)$$

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

Yet to say: there exists someone in CU that is smart

$$\exists \boldsymbol{x}\, At(x,CU) \wedge Smart(x)$$

## Examples of FOPC

- Brothers are siblings

  $\forall x, \forall y$ ***Brother(x,y)*** => ***Sibling(x,y)***

- One's mother is one's female parent

  $\forall m, \forall c$ ***Mother(c)*** = m $\Leftrightarrow$ ***(Female(m)*** $\wedge$ ***Parent(m,c))***

- “Sibling” is symmetric

  $\forall x, \forall y$ ***Sibling(x,y)*** $\Leftrightarrow$ ***Sibling(y,x)***

## Translating English to FOL

- Every gardener likes the sun.

  `(∀x) gardener(x) => likes(x,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))`

# 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))`

# 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)`

# 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)`

# Translating English to FOL

- 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))
  ```

# Inference in FOL

- ***KB*** $\vdash_i \alpha$ = sentence α can be derived from ***KB*** by procedure ***i***

i.e. deriving sentences from other sentences

- Soundness: ***i*** is sound if whenever ***KB*** $\vdash_i \alpha$, it is also true that ***KB*** $\models \alpha$

i.e. derivations produce only entailed sentences ***(no wrong inferences, but maybe not all inferences)***

- Completeness: ***i*** is complete if whenever ***KB*** $\models \alpha$, it is also true that ***KB*** $\vdash_i \alpha$

i.e. derivations can produce all entailed sentences (***all inferences can be made, but maybe some wrong extra ones as well)***

# Validity and satisfiability

- A sentence is **valid** if it is true in **all models**,
- e.g., ***True***, $A \vee \neg A$,  $A \Rightarrow A$,  $(A \wedge (A \Rightarrow B)) \Rightarrow B$

Validity is connected to inference via the following:
***KB*** $\models \alpha$ if and only if (***KB*** $\Rightarrow \alpha$) is valid

A sentence is **satisfiable** if it is true in **some model**
e.g., $A \vee B$,  C

A sentence is **unsatisfiable** if it is true in **no models**
e.g., $A \wedge \neg A$

Satisfiability is connected to inference via the following:
***KB*** $\models \alpha$ if and only if (***KB*** $\wedge \neg \alpha$) is unsatisfiable
(there is no model for which KB=true and  is false)

# Proof Methods in FOL

Major Families:

- GMP
- Reduction
- Resolution
- Forward chaining
- Backward chaining

Some Other inference tools:

Entailment/ Unification/

# Proof Methods in FOL

- GMP: Using the generalized form of Modus Ponense
- Reduction: Reduce all FOL sentences to propositional Calculus then use inference in propositional calculus
- Resolution – Refutation
  - Negate goal
  - Convert all pieces of knowledge into clausal form (disjunction of literals)
  - See if contradiction indicated by null clause $\square$ can be derived
- Forward chaining
  - Given P, $P \rightarrow Q$, to infer Q
  - P, match $L.H.S$ of
  - Assert Q from $R.H.S$
- Backward chaining
  - Q, Match $R.H.S$ of $P \rightarrow Q$
  - assert P
  - Check if P exists

# Universal instantiation (UI)

- Every instantiation of a universally quantified sentence is entailed by it:

$$\frac{\forall \boldsymbol{v}\, \alpha}{\text{Subst}(\{v/g\}, \alpha)}$$

for any variable $\boldsymbol{v}$ and ground term $\boldsymbol{g}$

- E.g., $\forall x$ $King(x) \wedge Greedy(x) \Rightarrow Evil(x)$ yields:

$King(John) \wedge Greedy(John) \Rightarrow Evil(John)$
$King(Richard) \wedge Greedy(Richard) \Rightarrow Evil(Richard)$
$King(Father(John)) \wedge Greedy(Father(John)) \Rightarrow Evil(Father(John))$

# Existential instantiation (EI)

- For any sentence α, variable $\boldsymbol{v}$, and constant symbol $\boldsymbol{k}$ that does not appear elsewhere in the knowledge base:

$$\frac{\exists \boldsymbol{v}\, \alpha}{\text{Subst}(\{v/k\}, \alpha)}$$

- E.g., $\exists x\ Crown(x) \wedge OnHead(x, John)$ yields:

$$Crown(C_1) \wedge OnHead(C_1, John)$$

provided $C_1$ is a new constant symbol, called a Skolem constant

# Unification

- $\forall x$ ***King*(*x*)** $\wedge$ ***Greedy*(*x*)** $\Rightarrow$ ***Evil*(*x*)**
- We can get the inference immediately if we can find a substitution θ such that *King(x)* and *Greedy(x)* match *King(John)* and *Greedy(y)*

θ = {x/John,y/John} works

- Unify(α,β) = θ if αθ = βθ

| p | q | θ |
|---|---|---|
| Knows(John,x) | Knows(John,Jane) | |
| Knows(John,x) | Knows(y,OJ) | |
| Knows(John,x) | Knows(y,Mother(y)) | |
| Knows(John,x) | Knows(x,OJ) | |

- Standardizing apart eliminates overlap of variables, e.g., Knows($z_{17}$,OJ)

# Unification

- We can get the inference immediately if we can find a substitution θ such that *King(x)* and *Greedy(x)* match *King(John)* and *Greedy(y)*

θ = {x/John,y/John} works

- Unify(α,β) = θ if αθ = βθ

| p | q | θ |
|---|---|---|
| Knows(John,x) | Knows(John,Jane) | {x/Jane}} |
| Knows(John,x) | Knows(y,OJ) | |
| Knows(John,x) | Knows(y,Mother(y)) | |
| Knows(John,x) | Knows(x,OJ) | |

- Standardizing apart eliminates overlap of variables, e.g., Knows($z_{17}$,OJ)

# Unification

- We can get the inference immediately if we can find a substitution θ such that *King(x)* and *Greedy(x)* match *King(John)* and *Greedy(y)*

θ = {x/John,y/John} works

- Unify(α,β) = θ if αθ = βθ

| p | q | θ |
|---|---|---|
| Knows(John,x) | Knows(John,Jane) | {x/Jane}} |
| Knows(John,x) | Knows(y,OJ) | {x/OJ,y/John}} |
| Knows(John,x) | Knows(y,Mother(y)) | |
| Knows(John,x) | Knows(x,OJ) | |

- Standardizing apart eliminates overlap of variables, e.g., Knows($z_{17}$,OJ)

# Unification

- We can get the inference immediately if we can find a substitution θ such that *King(x)* and *Greedy(x)* match *King(John)* and *Greedy(y)*

θ = {x/John,y/John} works

- Unify(α,β) = θ if αθ = βθ

| p | q | θ |
|---|---|---|
| Knows(John,x) | Knows(John,Jane) | {x/Jane} |
| Knows(John,x) | Knows(y,OJ) | {x/OJ,y/John} |
| Knows(John,x) | Knows(y,Mother(y)) | {y/John,x/Mother(John)} |
| Knows(John,x) | Knows(x,OJ) | |

- Standardizing apart eliminates overlap of variables, e.g., Knows($z_{17}$,OJ)

# Unification

- We can get the inference immediately if we can find a substitution θ such that *King(x)* and *Greedy(x)* match *King(John)* and *Greedy(y)*

θ = {x/John,y/John} works

- Unify(α,β) = θ if αθ = βθ

| p | q | θ |
|---|---|---|
| Knows(John,x) | Knows(John,Jane) | {x/Jane} |
| Knows(John,x) | Knows(y,OJ) | {x/OJ,y/John} |
| Knows(John,x) | Knows(y,Mother(y)) | {y/John,x/Mother(John)} |
| Knows(John,x) | Knows(x,OJ) | {fail} |

- Standardizing apart eliminates overlap of variables, e.g., Knows($z_{17}$,OJ)

# Unification

- To unify *Knows(John,x)* and *Knows(y,z)*,

  θ = {y/John, x/z } or θ = {y/John, x/John, z/John} or others…
- There are many possible unifiers for some atomic sentences. The first unifier is more general than the second.

- The UNIFY algorithm returns the most general unifier (MGU) that is unique up to renaming of variables. MGU makes the least commitment to variable values.

# The Unification Algorithm

•In order to match sentences in the KB, we need a routine.
•UNIFY(p,q) takes two atomic sentences and returns a substitution that makes them equivalent.
UNIFY(p,q)= θ where SUBST(θ,p)=SUBST(θ,q) θ is called a unifier.

```
function UNIFY(x, y, θ) returns a substitution to make x and y identical
   inputs: x, a variable, constant, list, or compound
           y, a variable, constant, list, or compound
           θ, the substitution built up so far

   if θ = failure then return failure
   else if x = y then return θ
   else if VARIABLE?(x) then return UNIFY-VAR(x, y, θ)
   else if VARIABLE?(y) then return UNIFY-VAR(y, x, θ)
   else if COMPOUND?(x) and COMPOUND?(y) then
       return UNIFY(ARGS[x], ARGS[y], UNIFY(OP[x], OP[y], θ))
   else if LIST?(x) and LIST?(y) then
       return UNIFY(REST[x], REST[y], UNIFY(FIRST[x], FIRST[y], θ))
   else return failure
```

# The Unification Algorithm

```
function UNIFY-VAR(var, x, θ) returns a substitution
   inputs: var, a variable
           x, any expression
           θ, the substitution built up so far

   if {var/val} ∈ θ then return UNIFY(val, x, θ)
   else if {x/val} ∈ θ then return UNIFY(var, val, θ)
   else if OCCUR-CHECK?(var, x) then return failure
   else return add {var/x} to θ
```