

## Propositional Logic: Syntax

Propositional logic consists of propositional symbols and the connectives  $\wedge$ ,  $\vee$ ,  $\neg$ ,  $\rightarrow$  (I like  $\rightarrow$  better than  $\Rightarrow$ ), and  $\leftrightarrow$ .

A *proposition* or propositional *sentence* can be formed as follows:

Every propositional symbol is a sentence.

If  $A$  is a sentence, then  $\neg A$  is a sentence.

If  $A_1$  and  $A_2$  are sentences, then so are:

$A_1 \wedge A_2$ , (and, conjunction)

$A_1 \vee A_2$ , (or, disjunction)

$A_1 \rightarrow A_2$ , and (implication)

$A_1 \leftrightarrow A_2$ . (equivalence)

A *literal* is a propositional symbol or its negation.

## Propositional Logic: Semantics

A *world* is the set of facts we want to represent.

An *interpretation* maps each propositional symbol to the world.

A sentence is *true* if its interpretation in the world is true.

A *knowledge base* is a set of sentences.

A world is a *model* of a KB if the KB is true for that world.

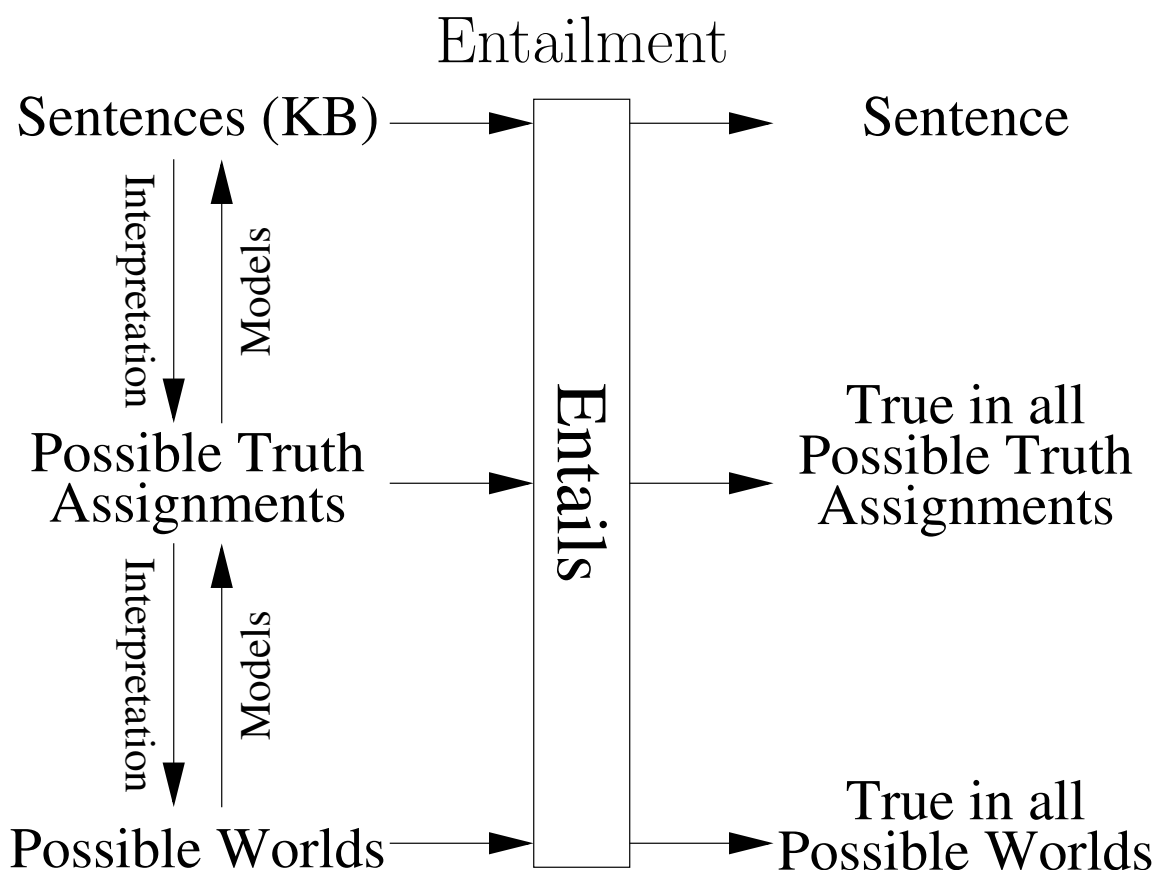
---

For convenience, a world/model can be thought of as a truth assignment to the symbols.

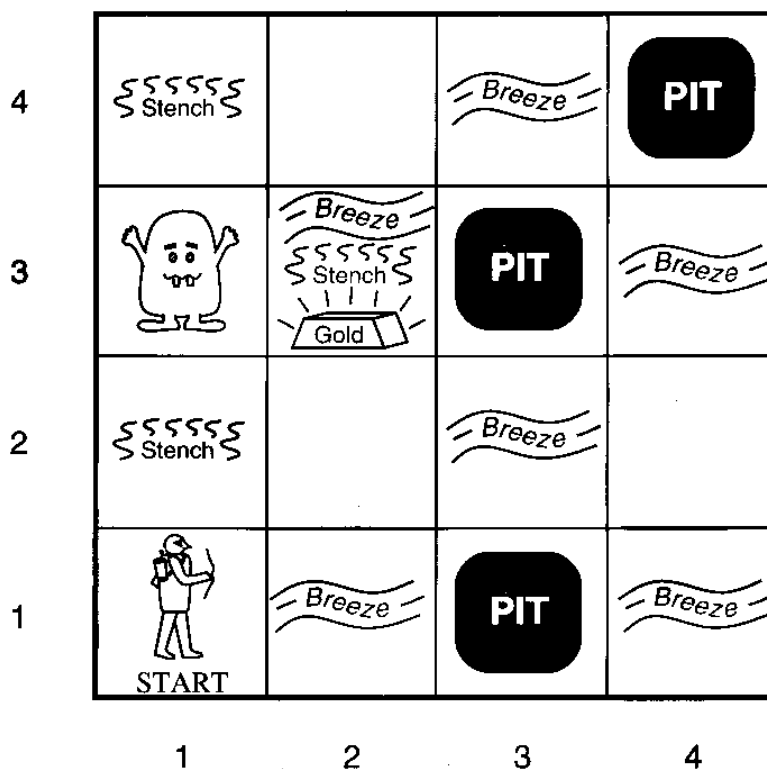
A sentence is *satisfiable* within a KB if the sentence is true in some model of the KB.

A sentence  $S$  is *entailed* by a KB if  $S$  is true in all models of the KB (denoted as  $KB \models S$ ), or equivalently, if  $KB \rightarrow S$  is *valid*.

*Logical inference* or *deduction* is concerned with producing entailed sentences from KBs.



## The Wumpus World



## Propositional Logic: Inference Rules

Let  $A$ ,  $B$ , and  $C$  be arbitrary sentences.

Modus Ponens From  $A \rightarrow B$   
and  $A$   

---

Infer  $B$

Unit Resolution From  $A \vee B$   
and  $\neg B$   

---

Infer  $A$

---

Resolution From  $A \vee B$   
and  $\neg B \vee C$   

---

Infer  $A \vee C$

An *inference procedure* uses inference rules to produce proofs. Denoted as  $KB \vdash S$ .

An inference procedure is *sound* if it cannot prove any unentailed sentence.

An inference procedure is *complete* if it can prove any entailed sentence.

## Propositional Logic: More on Resolution

Resolution applies to *conjunctive normal form*.

A KB is a conjunction of sentences.

Each sentence is a disjunction of literals.

$$(p \rightarrow q) \equiv (\neg p \vee q)$$

$$((p \wedge r) \rightarrow (q \vee s)) \equiv (\neg p \vee \neg r \vee q \vee s)$$

$$(p \rightarrow (q \wedge r))$$

$$\equiv ((p \rightarrow q) \wedge (p \rightarrow r))$$

$$\equiv ((\neg p \vee q) \wedge (\neg p \vee r))$$

Resolution is *refutation complete*. If a KB is in CNF, and if the KB is inconsistent, then resolution will infer inconsistent literals.

To deduce a sentence  $S$ , first temporarily add  $\neg S$  to the KB. Then, if resolution infers inconsistent literals, then  $\neg S$  is not satisfiable within the KB, which means that  $S$  is entailed.

## First-Order Logic: Syntax

First-order logic consists of predicates, functions, constant terms, and variable terms. First-order logic also uses the connectives  $\wedge$ ,  $\vee$ ,  $\neg$ ,  $\rightarrow$ , and  $\leftrightarrow$ , and the *quantifiers*  $\forall$  and  $\exists$ .

A *term* is:

- a constant or variable, or
- a function applied to a sequence of terms.

An *atomic sentence* or *atom* is:

- a predicate applied to a sequence of terms.

---

A first-order logic *sentence* can be formed as follows:

An atomic sentence is a sentence.

Connectives can be used in the usual way.

If  $A$  is a sentence and  $x$  is a variable, then:

$\forall x A$  is a sentence (universal quantifier),

and

$\exists x A$  is a sentence (existential quantifier).

A *well-formed formula* is a sentence in which all the variables are quantified.

## First-order logic: Semantics

The connectives  $\wedge$ ,  $\vee$ ,  $\neg$ ,  $\rightarrow$ , and  $\leftrightarrow$  are evaluated in the usual way.

$\forall x A$  is true if every substitution for  $x$  makes  $A$  true.

$\exists x A$  is true if at least one substitution for  $x$  makes  $A$  true.

---

An *interpretation* consists of objects in the world and mappings for terms and predicates.

A *ground term* is a term with no variables. Each ground term is mapped to an object.

Each predicate is mapped to a relation (think relational database).

A *ground atom* is an atomic sentence with no variables. A ground atom is *true* if the predicate's relation holds between the terms' objects.

## First-Order Logic: Inference Rules

Let  $A(x)$ ,  $B(y)$ , and  $C(z)$  be arbitrary unquantified propositions with logical variables.

Universal	From $\forall x A(x)$
Elimination	Infer $A(t)$ where $t$ is any term
Existential	From $\exists x A(x)$
Elimination	Infer $A(c)$ where $c$ is a new constant
Resolution (disjunctive)	From $\forall x, y B(y) \vee A(x)$ and $\forall x, z \neg A(x) \vee C(z)$
	Infer $\forall y, z B(y) \vee C(z)$
Resolution (implicative)	From $\forall x, y B(y) \rightarrow A(x)$ and $\forall x, z A(x) \rightarrow C(z)$
	Infer $\forall y, z B(y) \rightarrow C(z)$

## First-Order Logic: Example

### Knowledge Base

```
-parent(x,y) | -ancestor(y,z) | ancestor(x,z)
-parent(x,y) | ancestor(x,y)
-mother(x,y) | parent(x,y)
-father(x,y) | parent(x,y)
mother(Liz,Charley)
father(Charley,Billy)
```

To prove `ancestor(Liz,Billy)`

Refute `-ancestor(Liz,Billy)`

---

```
-parent(x,y) | -ancestor(y,z) | ancestor(x,z)
-ancestor(Liz,Billy)
```

```
-----
-parent(Liz,y) | -ancestor(y,Billy)
```

```
-mother(x,y) | parent(x,y)
-parent(Liz,y) | -ancestor(y,Billy)
```

```
-----
-mother(Liz,y) | -ancestor(y,Billy)
```

```
mother(Liz,Charley)
-mother(Liz,y) | -ancestor(y,Billy)
```

```
-----
-ancestor(Charley,Billy)
```

```
-parent(x,y) | ancestor(x,y)
-ancestor(Charley,Billy)
```

---

```
-parent(Charley,Billy)
```

```
-father(x,y) | parent(x,y)
-parent(Charley,Billy)
```

---

```
-father(Charley,Billy)
```

```
father(Charley,Billy)
-father(Charley,Billy)
```

---

```
contradiction
```

---

## Unification

To use the resolution inference rule, we need to be able to match atoms in sentences. This is called *unification*.

If successful, unification returns a *substitution*.

A substitution specifies values for variables that would make the two atoms identical.

```

function UNIFY-LISTS( $A, B, \theta$ )
  if  $A$  and  $B$  have different predicates/functions
    or have different numbers of arguments
  then return failure
  for  $i \leftarrow 1$  to number of arguments do
     $termA \leftarrow$   $i$ th argument of  $A$ 
     $termB \leftarrow$   $i$ th argument of  $B$ 
     $\theta \leftarrow$  UNIFY-TERMS( $termA, termB, \theta$ )
    if  $\theta =$  failure then return failure
  end for
  return  $\theta$ 

```

---

```

function UNIFY-TERMS( $A, B, \theta$ )
  while  $A$  is a variable and  $\theta[A]$  exists
    do  $A \leftarrow \theta[A]$ 
  while  $B$  is a variable and  $\theta[B]$  exists
    do  $B \leftarrow \theta[B]$ 
  if  $A = B$  then do nothing
  else if  $A$  is a variable then  $\theta[A] \leftarrow B$ 
  else if  $B$  is a variable then  $\theta[B] \leftarrow A$ 
  else if  $A$  or  $B$  is a constant then  $\theta \leftarrow$  failure
  else  $\theta \leftarrow$  UNIFY-LISTS( $A, B, \theta$ )
  return  $\theta$ 

```

## Resolution

**RESOLVE** assumes that sentences  $A$  and  $B$  are disjunctions represented as sets of literals. A literal is an atom or its negation.

**RESOLVE** assumes that sentences  $A$  and  $B$  have no variables with the same names.

**RESOLVE** returns all the sentences it infers.

```

function RESOLVE( $A, B$ )
  for each  $litA$  in  $A$  do
    for each  $litB$  in  $B$  do
      if one of  $litA$  and  $litB$  is negated, not both then
         $\theta \leftarrow$  UNIFY-LISTS( $litA, litB, \text{empty substitution}$ )
        if  $\theta \neq$  failure and has no recursive substs. then
           $A' \leftarrow$  apply substitution  $\theta$  to  $A$ 
           $litA' \leftarrow$  apply substitution  $\theta$  to  $litA$ 
           $B' \leftarrow$  apply substitution  $\theta$  to  $B$ 
           $litB' \leftarrow$  apply substitution  $\theta$  to  $litB$ 
           $C \leftarrow (A' - \{litA'\}) \cup (B' - \{litB'\})$ 
          add  $C$  to formulas inferred
    return formulas inferred
  
```