10-0: **Representation**

- Propositional Logic has several nice features
  - Lets us easily express disjunction and negation
    - "There is a pit in (1,1) or in (2,2)"
    - "There is not a pit in (1,1)"
    - This is hard to do in C/Java/Python - variables can only take on a single value.
    - There's no obvious way to *assign x* the value "3 or 4" or "some value less than 10".
  - Separates declarative knowledge from inference procedures
  - Compositional
    - The meaning of a sentence is a function of the meaning of its parts.

10-1: **Review of Propositional Logic**

- Sentences are composed of atomic terms conjoined by operators
  - $P_{1,1} \wedge B_{1,2}$
  - $\neg B_{2,2} \vee \neg P_{1,2}$
- Terms are either true or false.
- A model is an assignment of values to terms.
  - The set of possible worlds that make a sentence true
- A model satisfies a sentence if the sentence is true given the assignments in the model.

10-2: **Resolution**

- Inference can be done using DeMorgan's, Modus Ponens, And-elimination, etc
- Sound, but not necessarily complete.
- Also, search can be inefficient: there might be many operators that can be applied in a particular state.
- Luckily, there is a complete rule for inference that uses a single operator.
- This is called *resolution*.
  - $A \vee B$ and $\neg A \vee C$ allows us to conclude $B \vee C$.
  - A is either true or not true. If A is true, then C must be true.
  - if A is false, then B must be true.
  - This can be generalized to clauses of any length.

10-3: **Conjunctive Normal Form**

- Resolution works with disjunctions.
- This means that our knowledge base needs to be in this form.
- Conjunctive Normal Form is a conjunction of clauses that are disjunctions.
- $(A \vee B \vee C) \wedge (D \vee E \vee F) \wedge (G \vee H \vee I) \wedge ...$
- Every propositional logic sentence can be converted to CNF.

10-4: **CNF Recipe**

1. Eliminate equivalence
   - $A \Leftrightarrow B$ becomes $A \Rightarrow B \wedge B \Rightarrow A$
2. Eliminate implication
   - $A \Rightarrow B$ becomes $\neg A \vee B$
3. Move $\neg$ inwards using double negation and DeMorgan's
   - $\neg(\neg A)$ becomes A
   - $\neg(A \wedge B)$ becomes $(\neg A \vee \neg B)$
4. Distribute nested clauses
   - $(A \vee (B \wedge C))$ becomes $(A \vee B) \wedge (A \vee C)$

10-5: **Example**

- $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
- Eliminating equivalence produces:
  - $(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$
- Removing implication gives us:
  - $(\neg B_{1,1} \vee (P_{1,2} \vee P_{2,1})) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$

10-6: **Example**

- We then use DeMorgan's rule to move negation inwards:
  - $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$
- Finally, we distribute OR over AND:
  - $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$
- Now we have clauses that can be plugged into a resolution theorem prover. (can break ANDs into separate sentences)
- They're less readable by a human, but more computationally useful.

10-7: **Proof By Refutation**

- Once your KB is in CNF, you can do resolution by refutation.
  - In math, this is called proof by contradiction
- Basic idea: we want to show that sentence A is true.
- Insert $\neg A$ into the KB and try to derive a contradiction.

10-8: **Example**

- Notation:
  - R1: Robot1 is on.
  - R2: Robot 2 is on.

- ChR1: Robot 1 is at the charger.
- ChR2: Robot 2 is at the charger.
- lowR1: Robot 1 has low power.
- lowR2: Robot 2 has low power.
- moveChR1: Robot 1 should move onto the charger.
- moveChR2: Robot 2 should move onto the charger.

10-9: **Example**

- Encode Convert to CNF:
  - If robot 1 is off, then robot 2 is off.
  - If robot 2 is off, then it will be located on the charger, and if it is not off, then robot 2 will not be located on the charger.
  - If robot 1 is off, then it will be located on the charger, and if it is not off, then robot 1 will not be located on the charger.
  - If Robot 1 is on, and its power is low, then it should move to the charger.
  - Robot 1's power is low.
  - Robot 2 is on.
- Use resolution with refutation to prove that Robot 1 should move to the charger.

10-10: **Expressivity**

- We would like the sorts of structures that are useful in programming languages. In particular, we would like:
  - Objects: Wumpi, pits, gold, vacuum cleaners, etc.
  - Variables: how do we talk about objects without knowing their names?

10-11: **Expressivity**

- We would like the sorts of structures that are useful in programming languages. In particular, we would like:
  - Relations: These can include:
    - Unary relations (or properties): smelly(wumpus), shiny(gold), sleepy(student), etc.
    - Binary relations: brother-of(bart, lisa), holding(agent, gold), after(Tuesday, Monday)
    - *n*-ary relations: simpsons(homer, marge, bart, lisa)
    - These are sometimes called *predicates*
  - Functions: father-of(bart) = homer, fall-classes(student) = AI, etc.
- *First-order logic* gives us all of this.

10-12: **Models in first-order logic**

- Recall that a model is the set of "possible worlds" for a collection of sentences.
- In propositional logic, this meant truth assignments to facts.
- In FOL, models have objects in them.
- The *domain* of a model is the set of objects in that world.
- For example, the Simpsons model might have the domain
  - {Marge, Homer, Lisa, Bart, Maggie}
- We can then specify relations and functions between these objects
  - married-to(marge, homer), baby(maggie), father(bart) = homer

10-13: **Terms and sentences**

- A *term* is an expression that refers to a single object.
  - Bart, Lisa, Homer
  - We can also use functions as terms - Saxophone(Lisa) refers to the object that is Lisa's saxophone
- An *atomic sentence* consists of a predicate applied to terms
  - Brother-of(Lisa, Bart), Married(Homer, Marge), Married(Mother(Lisa), Father(Bart))
  - Plays(Lisa, Saxophone(Lisa))

10-14: **Terms and sentences**

- A Complex sentence uses logical connectives $\neg, \vee, \wedge, \Rightarrow, \Leftrightarrow$ to join atomic sentences.
  - $\neg BrotherOf(Homer, Bart)$,
  - $MotherOf(Lisa, Marge) \Rightarrow MotherOf(Bart, Marge)$
  - $Oldest(Bart) \vee Oldest(Lisa)$
- We can also use equality to relate objects: $homer = father(Bart)$

10-15: **Quantifiers and variables**

- Often, it's not enough to make a statement about particular objects. Instead, we want to make a statement about some or all objects.
  - "All of the Simpsons are yellow."
  - "At least one of the Simpsons is a baby."
  - Quantifiers allow us to do this.
  - $\forall$ is the symbol for universal quantification
    - It means that a sentence holds for every object in the domain.
    - $\forall x Simpson(x) \Rightarrow yellow(x)$

10-16: **Quantifiers and variables**

- $\exists$ is the symbol for existential quantification
  - It means that the sentence is true for at least one element in the domain.
  - $\exists x female(x) \wedge playsSaxophone(x)$
  - What would happen if I said $\exists x female(x) \Rightarrow playsSaxophone(x)$?

10-17: **Quantifiers**

- In general, $\Rightarrow$ makes sense with $\forall$ ($\wedge$ is usually too strong).
- $\wedge$ makes sense with $\exists$ ($\Rightarrow$ is generally too weak.)
- Some examples:
  - One of the Simpsons works at a nuclear plant.
  - All of the Simpsons are cartoon characters.
  - There is a Simpson with blue hair and a green dress.

- There is a Simpson who doesn't have hair.

10-18: **Nesting quantifiers**

- Often, we'll want to express more complex quantifications. For example, "every person has a mother"
  - $\forall x \exists y mother(x, y)$
  - Notice the scope - for each x, a different y is (potentially) chosen.
- What if we said $\exists y \forall x mother(x, y)$?

10-19: **Nesting quantifiers**

- Nesting quantifies is not a problem when nesting quantifiers of the same type.
- $\forall x \forall y\ brotherOf(x, y) \Rightarrow siblingOf(x, y)$ and $\forall y \forall x\ brotherOf(x, y) \Rightarrow siblingOf(x, y)$ are equivalent.
- We often write that as $\forall x, y\ brotherOf(x, y) \Rightarrow siblingOf(x, y)$

10-20: **Negation**

- We can negate quantifiers
  - $\neg \forall x\ yellow(x)$ says that it is not true that everyone is yellow.
  - $\exists x\ \neg yellow(x)$ has the same meaning - there is someone who is not yellow.
  - $\neg \exists x\ daughterOf(Bart, x)$ says that there does not exist anyone who is Bart's daughter.
  - $\forall x\ \neg daughterOf(Bart, x)$ says that for all individuals they are not Bart's daughter.
- In fact, we can use DeMorgan's rules with quantifiers just like with $\wedge$ and $\vee$.

10-21: **More examples**

- A husband is a male spouse
  - $\forall x, y\ husband(x, y) \Leftrightarrow spouse(x, y) \wedge male(x)$
- Two siblings have a parent in common
  - $\forall x, y\ sibling(x, y) \Leftrightarrow$
    $\neg(x = y) \wedge \exists p Parent(x, p) \wedge Parent(y, p)$
- Everyone who goes to Moe's likes either Homer or Barney (but not both)
  - $\forall x\ goesTo(moes, x) \Rightarrow$
    $(Likes(x, Homer) \Leftrightarrow \neg Likes(x, Barney))$

10-22: **More examples**

- Everyone knows someone who is angry at Homer.
  - $\forall x \exists y\ knows(x, y) \wedge angryAt(y, homer)$
- Everyone who works at the power plant is scared of Mr. Burns
  - $\forall x\ worksAt(PowerPlant, x) \Rightarrow scaredOf(x, burns)$

10-23: **Audience Participation**

- Everyone likes Lisa.

- Someone who works at the power plant doesn't like Homer. (both ways)
- Bart, Lisa, and Maggie are Marge's only children.
- People who go to Moe's are depressed.
- There is someone in Springfield who is taller than everyone else.
- When a person is fired from the power plant, they go to Moe's
- Everyone loves Krusty except Sideshow Bob
- Only Bart skateboards to school
- Someone with large feet robbed the Quickie-mart.

10-24: **Repr. Knowledge in FOL**

- We can use FOL to represent class/subclass information, causality, existence, and disjoint sets.
- Example: Let's suppose we are interested in building an agent that can help recommend music.
- We want it to be able to reason about musical artists, songs, albums, and genres.
- It would be tedious to enter every bit of information about every artist; instead, we'll enter some rules and let our agent derive entailed knowledge.

10-25: **Music example**

- $\forall x\ genre(x, Punk) \rightarrow genre(x, Rock)$ - subclass: all Punk songs are Rock songs.
- $member(JohnLennon, Beatles) \wedge member(PaulMcCartney, Beatles) \wedge member(GeorgeHarrison, Beatles) \wedge member(RingoStarr, B$
  $\forall x\ member(x, Beatles) \rightarrow x \in \{John, Paul, George, Ringo\}$ - exclusive membership: John, Paul, George, and Ringo are the Beatles.

10-26: **Music example**

- $performedBy(Beatles, WhiteAlbum)$ The WhiteAlbum is a Beatles album
- $\forall x, y, z\ member(x, y) \wedge performedBy(y, z) \rightarrow playedOn(x, z)$ if someone is a member of a group, and that group performed an album, then that person played on that album.

10-27: **Music example**

- $genre(HolidaysInTheSun, Punk)$ - "Holidays in the Sun" is a Punk song.
- $\forall x\ genre(x, Rock) \rightarrow likes(Bob, x)$ Bob likes all rock songs.
  - We should be able to infer that Bob will like "Holidays in the Sun"
- $\forall w, x, y, z\ likes(x, y) \wedge member(z, y) \wedge performedBy(z, w) \rightarrow likes(x, w)$ - If someone likes albums by a band Y, and Z is a member of band Y, then that person will like albums by person Z.

10-28: **Inference in Prop. logic**

- We talked about two basic mechanisms for performing inference in propositional logic:
  - Forward chaining: Begin with the facts in your KB. Apply inference rules until no more conclusions can be drawn.

- Backward chaining: Begin with a fact (or its negation) that you wish to prove. Find facts that will justify this fact. Work backwards until you find facts in your KB that support (contradict) the fact you wish to prove.

10-29: **Inference in FOL**

- Can we do the same sorts of inference with First-order logic that we do with propositional logic?
- Yes, with some extra details.
- Need to keep track of variable bindings (substitution)

10-30: **Inference in FOL**

- As with propositional logic, we'll need to convert our knowledge base into a canonical form.
- In the simplest approach, we can convert our FOL sentences to propositional sentences. We do this by removing quantification.
  - we leave in predicates for readability, but remove all variables.

10-31: **Removing quantifiers**

- Universal Instantiation: we can always substitute a ground term for a variable.
- This will typically be a term that occurs in some other sentence of interest.
- Choose a substitution for x that helps us with our proof.
  - $\forall x LivesIn(x, Springfield) \Rightarrow knows(x, Homer)$
  - Since this is true for all x, we can substitute $\{x = Bart\}$
  - $LivesIn(Bart, Springfield) \Rightarrow knows(Bart, Homer)$

10-32: **Removing quantifiers**

- Existential Instantiation: we can give a name to the object that satisfies a sentence.
  - $\exists x LivesIn(x, Springfield) \wedge knows(x, Homer)$
  - We know this must hold for at least one object. Let's call that object K.
  - $LivesIn(K, Springfield) \wedge knows(K, Homer)$
  - K is called a *Skolem constant*.
  - K must be unused - gives us a way of referring to an existential object.
- Once we've removed quantifiers, we can use propositional inference rules.

10-33: **Skolemization**

- Existential Instantiation: we can give a name to the object that satisfies a sentence.
  - Everybody loves someone
  - $\forall x \exists y Loves(x, y)$
  - Replace the existential variable with e skolem constant, does it mean what we want?
  - $\forall x Loves(x, A)$
  - What does this mean?

10-34: **Skolemization**

- If an existential quantifier is within the scope of a universal quantifier, we need to replace the existential variable with a function of the universal variable
  - $\forall x \exists y Loves(x, y)$
  - $\forall x, Loves(x, F(x))$
  - What does this mean? Is it what we want?

10-35: **Skolemization**

- What about a Universal Quantifier within an existential quantifier?
  - There is someone whom everyone loves
  - $\exists x, \forall y Loves(y, x)$

10-36: **Skolemization**

- What about a Universal Quantifier within an existential quantifier?
  - There is someone whom everyone loves
  - $\exists x, \forall y Loves(y, x)$
  - $forallyLoves(y, K)$
- No need for a Skolem function, skolem constant works just fine

10-37: **Propositionalization**

- We can replace every existential sentence with a Skolemized version.
- For universally quantified sentences, substitute in *every* possible substitution.
- This will (in theory) allow us to use propositional inference rules.
- Problem: very inefficient!
- This was the state of the art until about 1960.
- Unification of variables is much more efficient.

10-38: **Unification**

- The key to unification is that we only want to make substitutions for those sentences that help us prove things.
- For example, if we know:
  - $\forall x LivesIn(x, Springfield) \wedge WorksAt(x, PowerPlant) \Rightarrow knows(x, Homer)$
  - $\forall y LivesIn(y, Springfield)$
  - $WorksAt(MrSmithers, PowerPlant)$
- We should be able to conclude $knows(MrSmithers, Homer)$ directly.
- Substitution: $\{x/MrSmithers, y/MrSmithers\}$

10-39: **Generalized Modus Ponens**

- This reasoning is a generalized form of Modus Ponens.
- Basic idea: Let's say we have:
  - An implication of the form $P_1 \wedge P_2 \wedge ... \wedge P_i \Rightarrow Q$
  - Sentences $P'_1, P'_2, ..., P'_i$
  - a set of substitutions such that $P_1 = P'_1, P_2 = P'_2, ..., P'_n = P_n$
- We can then apply the substitution and apply Modus Ponens to conclude Q.
- this technique of using substitutions to pair up sentences for inference is called *unification*.

10-40: **Unification**

- Our inference process now becomes one of finding substitutions that will allow us to derive new sentences.
- The Unify algorithm: takes two sentences, returns a set of substitutions that unifies the sentences.
  - $WorksAt(x, PowerPlant), WorksAt(Homer, PowerPlant)$ produces $\{x/Homer\}$.
  - $WorksAt(x, PowerPlant), WorksAt(Homer, y)$ produces $\{x/Homer, y/PowerPlant\}$
  - $WorksAt(x, PowerPlant), WorksAt(FatherOf(Bart), y)$ produces $\{x/FatherOf(Bart), y/PowerPlant\}$
  - $WorksAt(x, PowerPlant), WorksAt(Homer, x)$ fails - x can't bind to both Homer and PowerPlant.

10-41: **Unification**

- This last sentence is a problem only because we happened to use x in both sentences.
- We can replace x with a unique variable (say x_{21}) is one sentence.
  - This is called standardizing apart.

10-42: **Unification**

- What if there is more than one substitution that can make two sentences look the same?
  - $Sibling(Bart, x), Sibling(y, z)$
  - can produce$\{Bart/y, x/z\}$ or $\{x/Bart, y/Bart, z/Bart\}$
- the first unification is more general than the second - it makes fewer commitments.
- We want to find the *most general unifier* when performing inference.
  - (This is the heuristic in our search.)

10-43: **Unification Algorithm**

- To unify two sentences, proceed recursively.
- If either sentence is a single variable, find a unification that binds the variable to a constant.
- Else, call unify in the first term, followed by the rest of each sentence.
  - $Sibling(x, Bart)$ and $Sibling(Lisa, y)$
  - We can unify $Sibling(x, Bart)$ and $Sibling(Lisa, y)$ with $\{x/Lisa, y/Bart\}$

10-44: **Forward Chaining**

- Basic idea: Begin with facts and rules (implications)
- Continually apply Modus Ponens until no new facts can be derived.

- Requires *definite clauses*
  - Implications with positive clauses in the antecedent
  - Positive facts

10-45: **Forward Chaining Algorithm**

-
```
while (1) :
   for rule in rules :
      if (can_unify(rule, facts)) :
        fire_rule(rule)
        assert consequent facts
  if (no rules fired) :
     return
```

10-46: **Example** The law says that it is a crime for an American to sell weapons to hostile nations. The country of Nono, an enemy of America, has some missiles. All of its missiles were sold to it by Colonel West, who is an American.

- Prove that West is a criminal.
  - It is a crime for an American to sell weapons to hostile nations.
  - 1.$American(x) \wedge Weapon(y) \wedge Hostile(z) \wedge Sells(x, y, z) \Rightarrow Criminal(x)$
  - Nono has missiles.
  - 2.$\exists x Owns(Nono, x) \wedge Missile(x)$
  - Use Existential Elimination to substitute M_1 for x
  - 3.$Owns(Nono, M_1)$, 4.$Missile(M_1)$

10-47: **Example**

- Prove that West is a criminal.
  - All Nono's missiles were sold to it by Colonel West.
  - 5.$Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$
  - Missiles are weapons.
  - 6.$Missile(x) \Rightarrow Weapon(x)$
  - An enemy of America is a hostile nation.
  - 7.$Enemy(x, America) \Rightarrow Hostile(x)$
  - West is American
  - 8.$American(West)$
  - Nono is an enemy of America
  - 9.$Enemy(Nono, America)$
- We want to forward chain until we can show all the antecedents of 1.

10-48: **Example**

- Algorithm: Repeatedly fire all rules whose antecedents are satisfied.
- Rule 6 matches with Fact 4. $\{x/M_1\}$. Add $Weapon(M_1)$.
- Rule 5 matches with 3 and 4. $\{x/M_1\}$. Add $Sells(West, M_1, Nono)$
- Rule 7 matches with 9. $\{x/Nono\}$. Add $Hostile(Nono)$
- Iterate again.
- Now we can match rule 1 with $\{x/West, y/M_1, z/Nono\}$ and conclude $Criminal(West)$.

10-49: **Basic Forward Chaining**

- Forward chaining is sound, since it uses Modus Ponens.
- Forward chaining is complete for definite clauses.
- Works much like BFS
- This basic algorithm is not very efficient, though.
  - Finding all possible unifiers is expensive
  - Every rule is rechecked on every iteration.
  - Facts that do not lead to the goal are generated.

10-50: **Backward Chaining**

- Basic idea: work backward from the goal to the facts that must be asserted for the goal to hold.
- Uses Modus Ponens (in reverse) to focus search on finding clauses that can lead to a goal.
- Also uses definite clauses.
- Search proceeds in a depth-first manner.

10-51: **Backward Chaining Algorithm**

```
BackwardChain(goals, subst)
  if goals is empty, return True
  goal = goals.dequeue
  foreach sentence in KB
     if (unify(consequent(sentence, goal)))
        (giving new subst, subst')
        if (BackwardChain(antecedent(goal)+goals,
                          subst')
            return True
  return False
```

10-52: **Backward Chaining Example**

1. $American(x) \wedge Weapon(x) \wedge Hostile(x) \wedge Sells(x, y, z) \Rightarrow Criminal(x)$
2. $\Rightarrow Owns(Nono, M_1)$
3. $\Rightarrow Missle(M_1)$
4. $Missle(y) \wedge Owns(Nono, y) \Rightarrow Sells(West, y, Nono)$
5. $Missle(z) \Rightarrow Weapon(z)$
6. $Enemy(x, America) \Rightarrow Hostile(x)$
7. $\Rightarrow American(West)$
8. $\Rightarrow Enemy(Nono, America)$

10-53: **Analyzing Backward Chaining**

- Backward chaining uses depth-first search.
- This means that it suffers from repeated states.
- Also, it is not complete.
- Can be very effective for query-based systems
- Most backward chaining systems (esp. Prolog) give the programmer control over the search process, including backtracking.

10-54: **Resolution**

- Recall Resolution in Propositional Logic:

- $(A \vee C) \wedge (\neg A \vee B) \Rightarrow (B \vee C)$
- Resolution in FOL works similarly.
- Requires that sentences be in CNF.

10-55: **Conversion to CNF**

- The recipe for converting FOL sentences to CNF is similar to propositional logic.
  1. Eliminate Implications
  2. Move ¬ inwards
  3. Standardize apart
  4. Skolemize Existential sentences
  5. Drop universal quantifiers
  6. Distribute ∧ over ∨

10-56: **CNF conversion example**

- Sentence: Everyone who loves all animals is loved by someone.
- Translation: $\forall x(\forall y Animal(y) \Rightarrow loves(x, y)) \Rightarrow (\exists y Loves(y, x))$
- Eliminate implication
- $\forall x(\neg\forall y \neg Animal(y) \vee loves(x, y)) \vee (\exists y Loves(y, x))$
- Move negation inwards
  - $\forall x(\exists y \neg(\neg Animal(y) \vee Loves(x, y))) \vee (\exists y Loves(y, x))$
  - $\forall x(\exists y \neg\neg Animal(y) \wedge \neg Loves(x, y))) \vee (\exists y Loves(y, x))$
  - $\forall x(\exists y Animal(y) \wedge \neg Loves(x, y))) \vee (\exists y Loves(y, x))$
- Standardize apart
- $\forall x(\exists y Animal(y) \wedge \neg Loves(x, y))) \vee (\exists z Loves(z, x))$

10-57: **CNF conversion example**

- Skolemize. In this case we need a *Skolem function*, rather than a constant.
- $\forall x(Animal(F(x)) \wedge \neg Loves(x, y))) \vee (Loves((G(x), x))$
- Drop universals
- $(Animal(F(x)) \wedge \neg Loves(x, F(x)))) \vee (Loves((G(x), x))$
- Distribute ∧ over ∨
- $(Animal(F(x)) \vee Loves(G(x), x)) \wedge (\neg Loves(x, F(x)))) \vee (Loves((G(x), x))$

10-58: **Resolution Theorem Proving**

- Resolution proofs work by inserting the negated form of the sentence to prove into the knowledge base, and then attempting to derive a contradiction.
- A *set of support* is used to help guide search
  - These are facts that are likely to be helpful in the proof.
  - This provides a heuristic.

10-59: **Resolution Algorithm**

```
sos = [useful facts]
usable = all facts in KB

do
   fact = sos.pop
   foreach fact in usable
       resolve fact with usable
   simplify clauses, remove duplicates and tautologies
   if a clause has no literals :
      return refutation found
until
  sos = []
```

10-60: **Resolution Example**

- 1.$\neg American(x) \vee \neg Weapon(y) \vee \neg Sells(x,y,z) \vee \neg Hostile(z) \vee Criminal(x)$
- 2.$\neg Missile(x) \vee \neg Owns(Nono,x) \vee Sells(west,x,Nono)$
- 3.$\neg Enemy(x,America) \vee Hostile(x)$
- 4.$\neg Missile(x) \vee Weapon(x)$
- 5.$Owns(Nono,M_1)$
- 6.$Missile(M_1)$
- 7.$American(West)$
- 8.$Enemy(Nono,America)$
- 9.$\neg Criminal(West)$ (added)

10-61: **Resolution Example**

- Resolve 1 and 9. Add 10.$\neg American(West) \vee \neg Weapon(y) \vee \neg Sells(West,y,z) \vee \neg Hostile(z)$
- Resolve 10 and 7. Add 11.$\neg Weapon(y) \vee \neg Sells(West,y,z) \vee \neg Hostile(z)$
- Resolve 11 and 4. Add 12.$\neg Missile(y) \vee \neg Sells(West,y,z) \vee \neg Hostile(z)$
- Resolve 12 and 6. Add 13.$Sells(West,M_1,z) \vee \neg Hostile(z)$
- Resolve 13 and 2. Add 14.$\neg Missile(M_1) \vee \neg Owns(Nono,M_1) \vee Hostile(Nono)$
- Resolve 14 and 6. Add 15.$\neg Owns(Nono,M_1) \vee \neg Hostile(Nono)$
- Resolve 15 and 5. Add 16$\neg Hostile(Nono)$
- Resolve 16 and 3. Add 17.$\neg Enemy(Nono,America)$.
- Resolve 17 and 8. Contradiction!

10-62: **Analyzing Resolution**

- Resolution is refutation complete - if a sentences is unsatisfiable, resolution will discover a contradiction.
- Cannot always derive all consequences from a set of facts.
- Can produce nonconstructive proofs for existential goals.
  - Prove $\exists x likes(x, Homer)$ will be proven, but without an answer for who x is.
- Can use full FOL, rather than just definite clauses.