CISS450 Lecture 8: Logic agents

Yihsiang Liow

November 15, 2022

Table of contents I

Logic I

- Readings: AIMA3 Chapter 7, AIMA4 Chapter 7.
- Up to this point, an agent "thinks" in the sense that it answers questions such as
  - Is this the goal state?
  - Is the cost of this action the best?
- Now we want agents to think logically, i.e., perform logical deductions.
- Therefore we need to formalize how humans think logically. If we can formalize the way humans think, then we can make software agents think.
- First the big picture ... an overview of logic ...

Logic II

- If I know the following facts
  - "if a bullet hit my heart I will die"
  - "if I fire my gun I will have one bullet less"
  - "if I turn a door knob right and the door is not locked, it opens"
  - "if I eat my energy level goes up"
  - ... and a million other facts ...

  can I deduce "it is safe to enter the room in front of me"?

Logic III

- The process through logic is to create "logic sentences" corresponding to statements about the world which are either true or false and then formalize the "deduction" process. And we then connect percepts and actions to "logic sentences".
- Example. Informally, if given

  {"if it's raining then the ground is wet", "it's raining"}

  it follows that

  "the ground is wet"

Logic IV

- Example. Translating to formal propositional logic:

$$\{\text{ItIsRaining} \rightarrow \text{GroundIsWet},\ \ \text{ItIsRaining}\} \models \text{GroundIsWet}$$

  where

  - "ItIsRaining", "GroundIsWet" are atomic propositional sentences called **propositional variables**
  - "ItIsRaining $\rightarrow$ GroundIsWet" is a **compound** or **complex propositional sentence**
  - "$\models$" is "it follows that", formally **entails**.

Logic V

- There might be other propositions which are not true and not in the above collection of propositions $\{\ldots\}$. The collection $\{\ldots\}$ is called the **knowledge base** of an agent (human or software or hardware). For instance there might be a proposition "PigsCanFly" in the system which is not in the knowledge base.
- Frequently propositions are created by engineers.
- When will a particular proposition be placed in the knowledge base? Usually through data from sensors, i.e., through percepts.

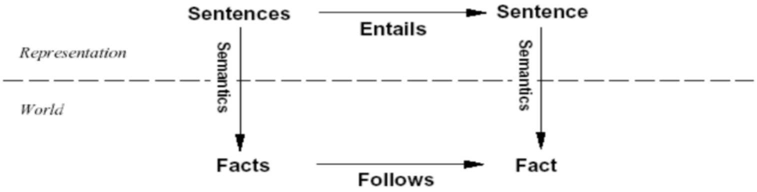
Logic VI

- Mathematically, to deduce a statement P from a collection of known statements $P_1, P_2, P_3, \ldots$, we say $P_1, P_2, P_3, \ldots$ **entails** P and we write

$$\{P_1, P_2, P_3, \ldots\} \models P$$

Logic VII

- Of course facts in the real world must correspond to propositions and deductions must correspond to logical deductions.



Logic VIII

- You can and should think of the above as a graph of sets of propositions where you are trying to create arcs ($\models$) connecting proposition to proposition. (A knowledge base is a set of propositions, but it is equivalent to one single proposition.)
- Basic notions in logic that helps in working out $\models$:
  - Equivalence
  - Tautology or validity
  - Contradiction
  - Satisfiability
  - Inference

Logic IX

- Logic = syntax + semantics
  - This is the same for any language.
- Syntax = What are the "valid" expressions / sentences?
- Semantics = What is the "meaning" of a valid expression?
- Note that syntax ≠ semantics: "Colorless green ideas sleep furiously" – Chomsky. Syntactically correct, but no meaning.
- Relation to agents: A logic agent will have a logic subsystem that will keep track of a knowledge base. The agent keeps adding logic sentences to the knowledge base from its percepts and actions. The logic subsystem will also make deductions and add new sentences to the knowledge base.

Logic X

- There are many classes of logic:
  - Propositional Logic
  - First-order Logic (or predicate logic)
  - Temporal logic: express time
  - Modal logic: express alternative worlds
  - Higher order logic: fancier quantifiers
- Example. Informal and formal predicate logic:

$$\{\text{All humans are mortal}, \text{Socrates is a human}\} \models \text{Socrates is mortal}$$
$$\{\forall x\,(\text{Human}(x) \to \text{Mortal}(x))\,, \text{Human}(\text{socrates})\} \models \text{Mortal}(\text{socrates})$$

Logic XI

- We will only talk about the first two categories of logic:
    - propositional logic
    - first order (or predicate) logic

  This set of notes focus on propositional logic.
- Propositional logic: Simpler. Involves true, false, and, or, not, etc.
- We start with propositional logic ... get ready ...

Syntax I

- **Syntax**: rules governing what sentences you can write down
- Related to how you want to represent knowledge
- In the case of **propositional logic** (**PL**), the set of all propositional logic sentences are generated by the following grammar rules (see CISS362 Automata theory or CISS445 Programming Languages for details):
    - $\textsc{True}$, $\textsc{False}$ are sentences
    - Propositional variables are sentences: P, Q, ...
    - If e, e' are sentences, then so are (e), $\neg e$, $e \wedge e'$, $e \vee e'$, $e \rightarrow e'$, $e \leftrightarrow e'$
- Propositional variables correspond to facts which are true or false in the real world. Example: or ItIsRaining or GroundIsWet or MonsterIsKilled.

Syntax II

- You create propositional variables to model facts which are either true or false and are relevant to solving your problem(s).
- Note that propositional variables are boolean variables. They can be substituted with TRUE or FALSE.
- **Connectives** (see truth table under semantics):

$\neg$:	models logical not
$\wedge$:	models logical and (also called **conjunction**)
$\vee$:	models logical or (also called **disjunction**)
$\rightarrow$:	models logical implication
$\leftrightarrow$:	models implication and implied by (also called **biconditional**)

Syntax III

- Propositional variable are **atomic** (i.e. not compound). For instance you should not create propositional variable JohnIsTallAndSkinny. Instead you should create two propositional variables JohnIsTall and JohnIsSkinny. Then

$$\text{JohnIsTallAndSkinny} = \text{JohnIsTall} \land \text{JohnIsSkinny}$$

Syntax IV

- Using formal grammar notation, if $P, Q, \ldots$ are propositional variables:

```
Sentence        ⟶ AtomicSentence | ComplexSentence
AtomicSentence  ⟶ TRUE | FALSE | P | Q | ...
ComplexSentence ⟶ ( Sentence )
                 |  Sentence ∧ Sentence
                 |  Sentence ∨ Sentence
                 |  Sentence → Sentence
                 |  Sentence ↔ Sentence
```

- Precedence rules:

$$\text{(highest)} \qquad \neg, \wedge, \vee, \rightarrow, \leftrightarrow \qquad \text{(lowest)}$$

Syntax V

- Association rules: For example

$$P \wedge Q \wedge R = (P \wedge Q) \wedge R = P \wedge (Q \wedge R)$$

Same for $\vee$. And

$$\neg\neg P = \neg(\neg P)$$

Syntax VI

- Example. Let P, Q, R be propositional variables.
  - $P \wedge \neg Q$ is a shorthand for $P \wedge (\neg Q)$
  - $P \vee Q \rightarrow Q \wedge \neg R$ is a shorthand for $(P \vee Q) \rightarrow (Q \wedge (\neg R))$
- Exercise. Fully parenthesize the following expressions according to the operator precedence conventions.
  - $P \wedge \neg Q \rightarrow (R \vee P) \wedge Q$ is a shorthand for
  - $\neg P \wedge Q \rightarrow \neg R \wedge \neg\neg P$ is a shorthand for

Syntax VII

- Example. Let P, Q, R be propositional variables. The following are valid propositional logic sentences
  - $P \wedge Q \wedge \neg R \rightarrow \neg P \wedge Q \vee \textsc{True}$
  - $P \wedge Q \wedge \neg R \leftrightarrow \neg P \wedge \neg Q$

  The following are not valid propositional logic sentences
  - $\wedge P$
  - $P \wedge Q \rightarrow$
  - $\leftrightarrow Q \neg R$
  - $P \vee Q \neg$

Syntax VIII

- Exercise. The following propositional logic describes the state of a 2-by-2 tic-tac-toe game. We are describing the valid states. There's no "time" involved. Three propositions corresponding to the 3 states of cell (0,0) are given.

$$\begin{aligned} N00 &= \text{“Cell (0,0) is not taken”} \\ X00 &= \text{“Cell (0,0) is taken by X”} \\ O00 &= \text{“Cell (0,0) is taken by O”} \end{aligned}$$

  - What other propositional variables do you need to express all the possible states of all the cells? (Named in the obvious way.)
  - Obviously one and only one of N00, X00, O00 can hold. How would you express that as a propositional sentence called C00?

Syntax IX

- Write a propositional sentence XWins that represents X is the winner.
- How would you write a propositional sentence Draw to represent the game has ended in a draw?

Syntax X

- In the tic-tac-toe example, note something important. Suppose the agent plays X in the tic-tac-toe game.
  - Some propositions are based on what is perceived by the agent: ("Cell (0,0) is taken by O")
  - Some propositions are based on an action of the agent itself: ("Cell (0,0) is taken by X")
  - some propositions are truth all the time ("Cell (0,0) is not taken xor taken by X xor taken by O").

  Because the first two types of propositions are based on "time", i.e., they are not true all the time, usually you need to include "time" in the propositions. More on this later.

Syntax XI

- A logic software library might work like this:

```
p = PropVar("john_is_tall")
q = PropVar("grass_is_orange")
r = AND(p, q)
s = OR(r, TRUE)
print(p); print(q); print(r); print(s)
```

Output:

```
john_is_tall
grass_is_orange
(john_is_tall & grass_is_orange)
(john_is_tall & grass_is_orange) | TRUE)
```

Semantics I

- **Semantics**: true or false value of a sentence after assigning true/false values to propositional variables
- This is with respect to a model (see later slides).
- However there are universal sentences
  - TRUE (as a sentence) is always assigned logical true value
  - FALSE (as a sentence) is always assigned logical false value
- There are rules for computing the true/false values for sentences once the values are given to propositional variables

Semantics II

- Truth tables:

P	Q	$P \land Q$	$P \lor Q$	$P \to Q$	$P \leftrightarrow Q$
FALSE	FALSE	FALSE	FALSE	TRUE	TRUE
FALSE	TRUE	FALSE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE	FALSE
TRUE	TRUE	TRUE	TRUE	TRUE	TRUE

P	$\neg P$
FALSE	FALSE
TRUE	TRUE

Semantics III

- Exercise. Complete the truth table of $P \wedge Q \rightarrow P \vee \neg Q$.

P	Q	$P \wedge Q$	$\neg Q$	$P \vee \neg Q$	$P \wedge Q \rightarrow P \vee \neg Q$
FALSE	FALSE				
FALSE	TRUE				
TRUE	FALSE				
TRUE	TRUE				

- Exercise. Write down the truth table of $P \leftrightarrow \neg P$.
- Exercise. Write down the truth table of $P \vee \neg Q \leftrightarrow \neg P \vee Q$.

Semantics IV

- In the above, basically I replaced propositional variables with TRUE or FALSE. You can think of this as **substitution**. For instance

  $$(P \wedge Q \vee \neg R)(Q = \text{TRUE}, R = \text{FALSE}) = (P \wedge \text{TRUE} \vee \neg \text{FALSE})$$

  Note that allow the case where not all propositional variables are replaced.

Semantics V

- Note that I use this notation for substitution:

$$e(P = \textsc{True}, Q = \textsc{False}, ...)$$

  Technical speaking in formal language/logic theory, the proper notation is

$$e[(P, Q, ...)/(\textsc{True}, \textsc{False}, ...)]$$

- Technically speaking it's also possible to substitute a propositional variable by an expression:

$$(P \wedge Q \vee \neg R)(Q = \textsc{True}, R = A \wedge B) = P \wedge \textsc{True} \vee \neg (A \wedge B)$$

Semantics VI

- Note that

  $$(P \wedge Q \vee \neg R)(Q = \text{TRUE}, R = \text{FALSE}) = (P \wedge \text{TRUE} \vee \neg \text{FALSE})$$

  means Q and R in $P \wedge Q \vee \neg R$ are substituted at the same time, not one after another.
  - Exercise. Can you think of an example where $e[X/x, Y/y]$ is not the same as $e[X/x][Y/y]$?

Models I

- Think of model as a "world" Formally, a **model** is an assignment of $\textsc{True}/\textsc{False}$ value to all the propositional variables.
  - Similar to substitution except that for for a model all propositional variable must be replaced by $\textsc{True}/\textsc{False}$
- Example. $\{P = \textsc{True}, Q = \textsc{True}, R = \textsc{False}\}$ is a model if P, Q, R are propositional variables. Another common notation: $\{P \to \textsc{True}, Q \to \textsc{True}, R \to \textsc{False}\}$
- If S is a sentence and m is a model, write $S(m)$ for S where each variable P of S is substituted by v if $P = v$ is in m.
  - Another notation: $\textsc{Subst}(m, S)$
  - Example. $(P \wedge Q)(\{P = \textsc{True}, R = \textsc{False}\}) = \textsc{True} \wedge Q$.

Models II

- Clearly

$$
\begin{aligned}
(S \wedge S')(m) &= S(m) \wedge S'(m) \\
(S \vee S')(m) &= S(m) \vee S'(m) \\
(\neg S)(m) &= \neg(S(m))
\end{aligned}
$$

  etc.

- Given a sentence S, if S **is true or holds in** model m if after substituting the variables in S with the values assigned in m and interpreting the symbol $\wedge$ as logical and, $\vee$ as logical not, ..., the resulting value is $\textsc{True}$. If that's the case I will write $S(m) = \textsc{True}$. Otherwise I write $S(m) = \textsc{False}$.

Models III

- $M(S)$ denotes the set of all models m such that $S(m) = \textsc{True}$. My notation: $\textsc{Models}(S)$.
- Note that frequently I will assume if m is a model in $\textsc{Models}(S)$, then m only contains variables appearing in S.
- Example.

$$\begin{aligned}\textsc{Models}(P \vee Q) = \{&\{P = \textsc{True}, Q = \textsc{True}\},\\ &\{P = \textsc{True}, Q = \textsc{False}\},\\ &\{P = \textsc{False}, Q = \textsc{True}\}\}\end{aligned}$$

Technically speaking if the system has propositional variables P, Q, R, I need to include R in the models as well.

Models IV

- Exercise. Assume P, Q, R are all the propositional variables.
    - What is $\textsc{Models}(P \vee Q \wedge \neg R)$?
    - What is $\textsc{Models}(P \rightarrow Q \wedge \neg R)$?
    - What is $\textsc{Models}(P \vee Q \rightarrow \neg Q \vee R)$?
    - What is $\textsc{Models}(P \vee Q \leftrightarrow \neg Q \vee R)$?

Equivalence I

- Let S, S' be sentences. S, S' are **equivalent**, and write

  $$S \equiv S'$$

  if they are true in the same models, i.e., for each model m,
  - if S holds in m, then S' holds in m
  - if S' holds in m, then S holds in m

  The above is the same as saying $\text{MODELS}(S) = \text{MODELS}(S')$.
- With $\equiv$ we can say two sentences are "the same".
- WARNING: There are two concepts of "sameness". If P is a propositional variable, then P and $P \wedge P$ are equivalent but different *as expressions*.

Equivalence II

- Example. Let P, Q, R be propositional variables.
  (a) $P \equiv P$
  (b) $P \wedge P \equiv P$
  (c) $P \wedge Q \equiv Q \wedge P$
  (d) $P \vee \neg P \equiv \textsc{True}$

  Prove the above statements.
  SOLUTION (a) For the P on the left-hand side of $P \equiv P$, $\textsc{Models}(P) = \{\{P = \textsc{True}, Q = \textsc{True}, R = \textsc{True}\}, \{P = \textsc{True}, Q = \textsc{True}, R = \textsc{False}\}, \{P = \textsc{True}, Q = \textsc{False}, R = \textsc{True}\}, \{P = \textsc{True}, Q = \textsc{False}, R = \textsc{False}\}\}$.
  The right-hand side of $P \equiv P$ is the same.

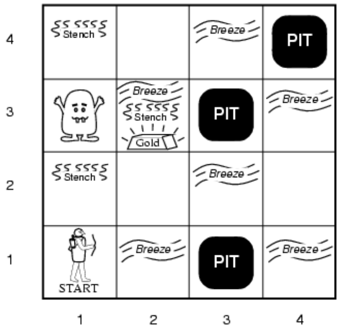
Equivalence III

- Standard equivalences:

Commutativity	$P \wedge Q \equiv Q \wedge P, \;\; P \vee Q \equiv Q \vee P$
Associativity	$(P \wedge Q) \wedge R \equiv P \wedge (Q \wedge R), \;\; (P \vee Q) \vee R \equiv P \vee (Q \vee R)$
Idempotent	$\neg\neg P \equiv P$
Contraposition	$P \rightarrow Q \equiv \neg Q \rightarrow \neg P$
Implication elimination	$P \rightarrow Q \equiv \neg P \vee Q$
Bicond elimination	$P \leftrightarrow Q \equiv (P \rightarrow Q) \wedge (Q \rightarrow R)$
De Morgan	$\neg(P \wedge Q) \equiv \neg P \vee \neg Q, \;\; \neg(P \vee Q) \equiv \neg P \wedge \neg Q$
Distributivity	$P \wedge (Q \vee R) \equiv (P \wedge Q) \vee (P \wedge R)$
	$P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$

Wumpus world I

- Wumpus world:



Wumpus world II

- Wumpus world:
  - Hunt the Wumpus – 1973 text-based adventure game developed by Gregory Yob.
  - Genesis of Wumpus by Gregory Yob from Artari Archives: `https://www.atariarchives.org/bcc1/showpage.php?page=247`.
  - See also `https://en.wikipedia.org/wiki/Hunt_the_Wumpus`.
  - There are many variations: `https://www.youtube.com/watch?v=xGVOw8gXl6Y`.
  - Wumpus world – introduced by AI researcher Michael Genesereth as a testbed for AI systems. Based on the Hunt the Wumpus.

Wumpus world III

- Environment
  - Cave with 4×4 rooms
  - Each square is labeled (row, column) where row and column are in [1..4].

Wumpus world IV

- Agent
  - Start at $(1, 1)$ which is safe and facing right.
  - Can move forward, turn left 90°, turn right 90°.
  - Has one arrow. Can shoot arrow in direction agent is facing. Fired arrow travels in given direction and either kills Wumpus (in arrow's path) or hit a wall.
  - Dies if enters room with live wumpus
  - Dies if enters room with pit
  - Can pick up (a heap of) gold
  - Can exit game by climbing out at (1,1).
  - Game ends once agent climb out at (1,1).
  - Actions: FORWARD, TURNLEFT, TURNRIGHT, SHOOT, GRAB, CLIMB.

Wumpus world V

- Sensors
  - In sq containing Wumpus and adj sq, agent perceives stench
  - In sq adj pit, agent perceives breeze. (Adj mean horizontally or vertically and not diagonally.)
  - In sq with Gold, agent perceives glitter
  - If agent walks into wall, perceives bump
  - If Wumpus dies, agent perceives scream
- Percept is [stench sensor data, breeze sensor data, glitter sensor data, bump sensor data, scream sensor data].

Wumpus world VI

- Other objects
    - Other objects (pit, gold, wumpus) are stationary and do not move.
    - 1 wumpus
    - 1 heap of gold
    - Each sq other than (1,1) has a probability $= 0.2$ of being a pit

Wumpus world VII

- Performance measure:
  - Gold: $+1000$
  - Death (eaten by Wumpus or fell into pit): -1000
  - Each action: -1
  - Use arrow: -10

Wumpus world VIII

- Propositional Variables:
    - $P_{i,j}$ = "Pit at (i,j)" (note there are 16 such variables)
    - $B_{i,j}$ = "Breeze at (i,j)"
    - $S_{i,j}$ = "Stench at (i,j)"
    - $W_{i,j}$ = "Wumpus at (i,j) (dead or alive)"
    - $GL_{i,j}$ = "Glitter at (i,j)"
    - $GO_{i,j}$ = "Gold at (i,j)"
- Notice that these are atomic in the sense that you can't simplify them. For instance "There is a breeze at $(2,2)$ and there is a glitter at $(3,4)$" is not atomic.

Wumpus world IX

- The initial knowledge base K:
  - Since $(1,1)$ is always safe, the K has
    - $\neg W_{1,1}$
    - $\neg P_{1,1}$
  - Since there is a breeze next to a pit, K has
    - $B_{1,1} \leftrightarrow (P_{1,2} \vee P_{2,1})$
    - $B_{2,1} \leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$
    - etc.

Wumpus world X

Note that these are true "regardless", which is another way of saying they are in K. Note that $P_{2,2}$ is not in K at the beginning, because the agent does not know whether there's a pit at (2,2) or not. If the agent arrives at a sq adj to (2,2) and the sensors does not pick up a breeze, then the agent can put $\neg P_{2,2}$ in K.

- As the agent walks in the wumpus world and collect sensor data, more sentences are placed in the KB K.
- Later we'll see that we have to incorporate "time".

Wumpus world XI

- Exercise:
  - $S_{3,1} \leftrightarrow$ ______________________________
  - What is the propositional sentence (in terms of above propositional variables) for "There is at least one wumpus?"
  - What does the proposition $\neg W_{1,1} \vee W_{1,2}$ mean?
  - What is the propositional sentence that says "There is at most one wumpus in $(1,1),(1,2),(2,1),(2,2)$?

Satisfiability I

- Recall: we are interested in making deductions, i.e. want to know when an entailment

$$K \models S$$

  holds.
- Entailment is a relation between propositions. So far you have seen one relation on propositions: equivalence, i.e., $\equiv$.
- Now we'll talk about
  - Satisfiability
  - Tautologies and validity
  - Entailment

Satisfiability II

- A sentence S is **satisfiable** if there is a model m such that S holds in m, i.e., $S(m) = \textsc{True}$, i.e., if $\textsc{Models}(S) \neq \emptyset$.
- A sentence S is not satisfiable if $\textsc{Models}(S) = \emptyset$.
- A fundamental concept in CS. Something that all CS people ought to know.
- Example. $P \vee \neg Q$ is satisfiable since it holds for $\{P = \textsc{True}, Q = \textsc{False}\}$.
- Example. $P \wedge (Q \vee R) \wedge \neg P$ is not satisfiable.
- Most problems in computer science are satisfiability problems.
  - CSP
  - $3\mathrm{SAT}$: Satisfiability for 3CNF proposition (see later)

Satisfiability III

- Exercise. Which of the following are satisfiable and which are not.
  - $(P \to Q) \land \neg P$
  - $P \to (Q \land \neg P)$
  - $(P \to Q) \land P$
  - $(P \to Q) \land \neg Q$
  - $(P \to Q) \land Q$
  - $(P \to Q) \land (P \to \neg Q) \land P$
- Exercise. What is the simplest propositional sentence you can think of that is not satisfiable?

Tautologies and validity I

- A sentence S is **valid** or is a **tautology** if S holds for all models.
- Example. Prove that $P \to P$ is a tautology
  SOLUTION. The following is the truth table of $P \to P$.

P	$P \to P$
FALSE	TRUE
TRUE	TRUE

  Hence $P \to P$ is a tautology.
- When proving S is a valid or is a tautology there's no point in including a variable if it does not appear in S.

Entailment I

- Let S, S' be sentences. We say S **entails** S' and write $S \models S'$ if for every model m where S is true, S' is also true in m.
- In other words $S \models S'$ if whenever $S(m)$ is $\textsc{True}$, then $S'(m)$ is also $\textsc{True}$, i.e.,

$$\textsc{Models}(S) \subseteq \textsc{Models}(S')$$

- If S does not entail S', we write $S \not\models S'$.

Entailment II

- Note that the brute force approach to prove

$$S \models S'$$

is to test all possible models with propositional variables appearing in S and S' (Obviously variables that do not appear in S or S' are not relevant.) If S and S' has a total of n variables, then there are 2^n models to test – exponential runtime!

Entailment III

- DEDUCTION THEOREM. $S \models S'$ iff $S \to S'$ is valid (or a tautology).

Proof. Recall that $S \to S' \equiv \neg S \vee S'$.

($\Rightarrow$): Let m be any model containing variables in S, S'. We want to show $(S \to S')(m) = \text{TRUE}$.

- Suppose $S(m) \equiv \text{TRUE}$. Since $S \models S'$, $S'(m) \equiv \text{TRUE}$.
- Suppose $S(m) = \text{FALSE}$. Then

$$(S \to S')(m) \equiv \neg\text{FALSE} \vee S'(m) \equiv \text{TRUE}$$

Entailment IV

In both cases $S \to S'$ is true. This holds for any model m containing variables in S, S'. Therefore $S \to S'$ is a tautology.
($\Leftarrow$): Let m be a model with variables in S, S'. Suppose m is a model for S, i.e., $S(m) = \text{TRUE}$. Since $S \to S' \equiv \neg S \vee S'$ is a tautology, we get

$$\neg\text{TRUE} \vee S'(m) \equiv \text{TRUE}$$

Therefore $S'(m) = \text{TRUE}$, i.e., m is a model for S'. This holds for any model m containing variables in S, S'. Therefore $S \models S'$. $\square$

Entailment V

- Note that if S, S' are sentences, then

$$S \equiv S'$$

if and only if

$$S \models S' \text{ and } S' \models S$$

which is the same as saying $\text{Models}(S) = \text{Models}(S')$.

- Therefore a logical equivalence gives an inference since equivalences are stronger than inferences. For instance since $P \wedge Q \equiv Q \wedge P$, we have

$$P \wedge Q \models Q \wedge P$$

Entailment VI

- Exercise. Let S, S' be sentences. Is it true that if S is a tautology, then $S \models S'$?
- Exercise. Let S, S' be sentences. Is it true that if S' is a tautology, then $S \models S'$?
- Exercise. Let S be a sentence. Suppose $S' \models S$ for any sentence S. What can you say about S?
- Exercise. Is entailment transitive? In other words suppose $S \models S'$ and $S' \models S''$ where S, S', S'' are sentences, then is it true that $S \models S''$?

Entailment VII

- Quick summary/comparison:

$S \equiv S'$	$\textsc{Models}(S) = \textsc{Models}(S')$
S is satisfiable	$\textsc{Models}(S)$ is not empty
S is a tautology or is valid	$S \equiv \textsc{True}$
$S \models S'$	$\textsc{Models}(S) \subseteq \textsc{Models}(S')$

Knowledge base I

- What if you need several sentences to make a conclusion?
- For instance suppose if you need both S, S' to be true in order to conclude that S'' is true. This is the same as

$$S \wedge S' \models S''$$

- However for computational purposes, it's frequently convenient to keep S and S' separate and write

$$\{S, S'\} \models S''$$

- Just need to adjust some definitions.

Knowledge base II

- **Knowledge base (KB)**: a set of sentences
  - Usually domain specific
  - Example: Knowledge base derived from mining pictures using facial recognition techniques
- Let m be a model and K be a knowledge base.
  - Then m is a model **for** K or K **holds** in m if every sentence in K is true in m. If that's the case, I'll write $K(m) = \textsc{True}$.
- $\textsc{Models}(K)$ is the set of all models of K.
- A knowledge base K **entails** S, and we write

$$K \models S$$

if $\textsc{Models}(K) \subseteq \textsc{Models}(S)$.

Knowledge base III

- Think of knowledge base as a collection of propositions in the agent's (software or hardware or human) brain that the agent perceives as truths.
- As more info is received (through percept), new propositions are added. The agent will also ask the logic subsystem to a proposition is true in order to achieve a goal.
- The knowledge base and the ability to prove satisfiability, validitiy, equivalence, entailment, etc. are part of the logic system in the agent.

Knowledge base IV

- Common names for the two most basic operations on knowledge base:
    - `TELL`: Add sentence to knowledge base. (Example: I see an open door)
    - `ASK`: Ask if a sentence follows from the knowledge base. (Example: Is it true that opening the door is safe?)

A software library for propositional logic might work like this:

```
KB = KnowledgeBase()
G = PropVar("GroundIsWet")
R = PropVar("ItIsRaining")
P = IMPLIES(R, G)
KB.tell(R)
KB.tell(P)
flag = KB.ask(G)
```

Model checking

- **Model checking** = Prove entailment

$$K \models S$$

  by going through all possible models. In other words prove $K \models S$ by using only the definition of entailment.
- Model checking is sound and complete (because it uses definition). But if K and S has n propositional variables, runtime is $O(2^n)$.
- The following algorithm `TT_ENTAILS`(K, S) returns TRUE iff $K \models S$. It assigns TRUE/FALSE value to more and more propositional variables to a partial model until a full model is created (think of CSP). Then substitution is performed using that full model. It then goes on to the next model.

Model checking

```
ALGORITHM: TT_ENTAILS:
INPUTS: KB - knowledge base
        S  - propositional sentence
OUTPUT: TRUE iff KB entails S

symbols = propositional vars of KB and S
return TT_CHECK_ALL(KB, S, symbols, {})

ALGORITHM: TT_CHECK_ALL(KB, S, symbols, model):
if symbols is empty:
    if KB(model) is TRUE:
        return S(model)
    else:
        return TRUE
else:
    P = first propositional var in symbols
    rest = symbols with P removed
    modelPT = model ∪ {P = TRUE}
    modelPF = model ∪ {P = FALSE}
    return TT_CHECK_ALL(KB, S, rest, modelPT)
    ∧ TT_CHECK_ALL(KB, S, rest, modelPF)
```

Model checking I

- Example partial trace: Let $K = \{X, X \wedge Y\}, S = Y \vee Z$. Think of K as $K = X \wedge (X \wedge Y)$. Now call `TT_ENTAILS`$(X \wedge (X \wedge Y), Y \vee Z)$.
  1. `TT_ENTAILS`$(X \wedge (X \wedge Y), Y \vee Z)$:
     - `symbols` is set to $\{X, Y, Z\}$.
     - Call `TT_CHECK_ALL`$(X \wedge (X \wedge Y), Y \vee Z, \{X, Y, Z\}, \{\})$.

Model checking II

2. TT_CHECK_ALL$(X \wedge (X \wedge Y), Y \vee Z, \{X, Y, Z\}, \{\})$:
   - P is set to X, rest is set to $\{Y, Z\}$
   - modelPT is set to $\{X = \text{True}\}$
   - modelPF is set to $\{X = \text{False}\}$
   - Call
     TT_CHECK_ALL$(X \wedge (X \wedge Y), Y \vee Z, \{Y, Z\}, \{X = \text{True}\})$
     and
     TT_CHECK_ALL$(X \wedge (X \wedge Y), Y \vee Z, \{Y, Z\}, \{X = \text{False}\})$
     and return $\wedge$ of their return values

Model checking III

3. TT_CHECK_ALL$(X \wedge (X \wedge Y), Y \vee Z, \{Y, Z\}, \{X = \textsc{True}\})$:
    - P is set to Y, rest is set to $\{Z\}$
    - modelPT is set to $\{X = \textsc{True}, Y = \textsc{True}\}$
    - modelPF is set to $\{X = \textsc{True}, Y = \textsc{False}\}$
    - Call TT_CHECK_ALL$(X \wedge (X \wedge Y), Y \vee Z, \{Z\}, \{X = \textsc{True}, Y = \textsc{True}\})$ and TT_CHECK_ALL$(X \wedge (X \wedge Y), Y \vee Z, \{Z\}, \{X = \textsc{True}, Y = \textsc{False}\})$ and return AND of their return values

Model checking IV

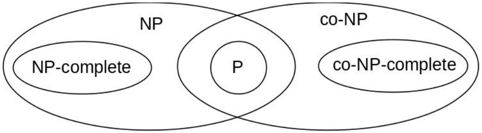
4. `TT_CHECK_ALL`$(X \wedge (X \wedge Y), Y \vee Z, \{Z\}, \{X = \text{True}, Y = \text{True}\})$:
   - P is set to Z, `rest` is set to $\{\}$
   - `modelPT` is set to $\{X = \text{True}, Y = \text{True}, Z = \text{True}\}$
   - `modelPF` is set to $\{X = \text{True}, Y = \text{True}, Z = \text{False}\}$
   - Call `TT_CHECK_ALL`$(X \wedge (X \wedge Y), Y \vee Z, \{\}, \{X = \text{True}, Y = \text{True}, Z = \text{True}\})$ and `TT_CHECK_ALL`$(X \wedge (X \wedge Y), Y \vee Z, \{\}, \{X = \text{True}, Y = \text{True}, Z = \text{False}\})$ and return AND of their return values

Model checking V

5. `TT_CHECK_ALL`$(X \wedge (X \wedge Y), Y \vee Z, \{\}, \{X = \text{TRUE}, Y = \text{TRUE}, Z = \text{TRUE}\})$:
    - $(X \wedge (X \wedge Y))(\{X = \text{TRUE}, Y = \text{TRUE}, Z = \text{TRUE}\})$ is TRUE. Return $(Y \vee Z)(\{X = \text{TRUE}, Y = \text{TRUE}, Z = \text{TRUE}\}) = \text{TRUE}$
6. Etc.

Model checking VI

- If K and S has n propositional variables, then
  - Time $= O(2^n)$
  - Space $= O(n)$ (the depth first length)
- Complete and sound
- All inference algorithms for propositional inference are co-NP-complete. It's believed:



Inference I

- Recall: There are basically three methods to check if $K \models S$:
  - Brute force model checking using truth table – DONE
  - Inference rules
  - Convert to inverse SAT problem (SAT $=$ satisfiability)
- Model checking is too slow. Now for the next two which uses "inference rules/laws".
- Given a knowledge base K and a sentence S, to **infer** S from K is to decide if $K \models S$ using an algorithm.
  - "Decide" means to return TRUE or FALSE.
- Difference between entails and infer: infer is entailment using an algorithm (the inference algorithm). For us, the algorithm will usually be based on inference rules/laws (see below).

Inference II

- If an entailment is computable using an inference algorithm A, for emphasize, I might write

$$K \vdash_A S$$

- Let A be an inference algorithm.
    - A is **sound** if A only infer sentences entailed by the KB, i.e.,

    $$(K \vdash_A S) \implies (K \models S)$$

    - A is **complete** if A can infer any sentences that is entailed by the KB, i.e.,

    $$(K \models S) \implies (K \vdash_A S)$$

Inference III

- Example. The inference algorithm XYZ works as follows: "Given K and B, to check if $K \vdash_{XYZ} S$, check if $K = P \wedge Q$ where P and Q are propositional variables, and $S = P$, return $\textsc{True}$. Otherwise return $\textsc{False}$.
    - Is A sound?
    - Is A complete?

Propositional theorem proving I

- **Theorem proving** = use inference rules to prove entailment. In other words no model checking. An inference is one step (usage of one inference rule). A proof usually involves more than one inference step.

Inference rules I

- Inference rules are written in this form:

$$\frac{P \rightarrow Q, \;\; P}{\therefore Q}$$

- The above notation means

$$\{P \rightarrow Q, \;\; P\} \vdash_{\text{inference rules}} Q$$

i.e.,

  - The entailment $\{P \rightarrow Q, \;\; P\} \models Q$ holds and
  - This is one of the standard “inference rules”

Inference rules II

- **Modus ponens or implication elimination**:

$$\frac{P \to Q, \;\; P}{\therefore Q}$$

- I'll write "elim" for "elimination".
- Example:

$$\frac{GL_{2,3} \wedge GL_{2,1} \wedge GL_{1,2} \wedge GL_{3,2} \to GO_{2,2}, \;\; GL_{2,3} \wedge GL_{2,1} \wedge GL_{1,2} \wedge GL_{3,2}}{GO_{2,2}}$$

- Exercise. Using a truth table, show that $\{P \to Q, P\} \models Q$, i.e., $(P \to Q) \wedge P \models Q$.

Inference rules III

- **Modus tollens or contrapositive elim**:

$$\frac{P \to Q, \neg Q}{\therefore \neg P}$$

- **Hypothetical syllogism or implication transitivity**:

$$\frac{P \to Q, \;\; Q \to R}{\therefore P \to R}$$

- **Disjunctive syllogism or Or-elim**:

$$\frac{P \lor Q, \;\; \neg P}{\therefore Q}$$

Inference rules IV

- Exercise. Look at modus tollens or contrapositive elimination. Using a truth table, prove that $P \to Q, \neg Q \models \neg P$, i.e., $(P \to Q) \wedge (\neg)Q \models \neg P$.

Inference rules V

- **And-elim**:

$$\frac{P \wedge Q}{\therefore P}$$

- **Or-intro(duction)**:

$$\frac{P, \;\; Q}{\therefore P \vee Q}$$

- Exercise. Using truth tables, prove
  - $P \wedge Q \models P$
  - $P \wedge Q \models P \vee Q$
  - Look at modus ponens. You have already shown $(P \to Q) \wedge P \models Q$ using a truth table. Now prove it using an equivalence and the and-elimination.

Inference rules VI

- Exercise. Look at modus tollens or contrapositive elimination again. You have shown $(P \rightarrow Q) \wedge (\neg Q) \models \neg P$ using a truth table. Now prove it without using a truth table.

Inference rules VII

- **Unit resolution**:

$$\frac{P \vee Q, \quad \neg Q}{\therefore P}$$

- **Binary resolution**:

$$\frac{P \vee Q, \quad \neg Q \vee R}{\therefore P \vee R}$$

- **Full resolution**:

$$\frac{P_1 \vee P_2 \vee \cdots P_i \vee Q \vee P_{i+1} \cdots P_m, \quad R_1 \vee R_2 \vee \cdots R_i \vee \neg Q \vee R_{i+1} \cdots R_n}{P_1 \vee \cdots \vee P_m \vee R_1 \vee \cdots \vee R_n \vee}$$

Inference rules VIII

- Resolution rules are very useful (later). INTUITION: Think of resultion rule as converting

  $$(\ldots \vee \ldots) \wedge (\ldots \vee \ldots)$$

  into

  $$(\ldots \vee \ldots)$$

- Exercise. Look at the unit resolution law. Using a truth table, show that $(P \vee Q) \wedge \neg Q \models P$. Next prove it using laws you know to hold.

Example: Wumpus world I

Given

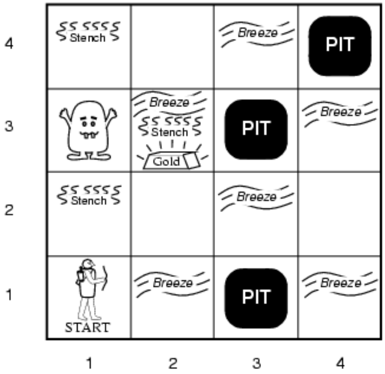
$R_1 : \neg P_{1,1}$

$R_2 : B_{1,1} \leftrightarrow (P_{1,2} \vee P_{2,1})$

$R_3 : B_{2,1} \leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$

$R_4 : \neg B_{1,1}$ i.e., no breeze at $(1,1)$)

$R_5 : B_{2,1}$ i.e., breeze at $(2,1)$)



- Prove $\neg P_{1,2} \wedge \neg P_{2,1}$ using inference rules.

Example: Wumpus world II

- SOLUTION. I need to show

$$\{R_1, R_2, R_3, R_4, R_5\} \models \neg P_{1,2} \land \neg P_{2,1}$$

  - Biconditional elimination on R_2:

$$\frac{R_2 : B_{1,1} \leftrightarrow (P_{1,2} \lor P_{2,1})}{R_6 : (B_{1,1} \rightarrow (P_{1,2} \lor P_{2,1})) \land ((P_{1,2} \lor P_{2,1}) \rightarrow B_{1,1})}$$

  - And-elimination on R_6:

$$\frac{R_6 : (B_{1,1} \rightarrow (P_{1,2} \lor P_{2,1})) \land ((P_{1,2} \lor P_{2,1}) \rightarrow B_{1,1})}{R_7 : (P_{1,2} \lor P_{2,1}) \rightarrow B_{1,1}}$$

Example: Wumpus world III

- Contraposition on R_7:

$$\frac{R_7 : (P_{1,2} \vee P_{2,1}) \rightarrow B_{1,1}}{R_8 : \neg B_{1,1} \rightarrow \neg(P_{1,2} \vee P_{2,1})}$$

- Modus Ponens on R_8, R_4:

$$\frac{R_8 : \neg B_{1,1} \rightarrow \neg(P_{1,2} \vee P_{2,1}), \quad R_4 : \neg B_{1,1}}{R9 : \neg(P_{1,2} \vee P_{2,1})}$$

- De Morgan on R_9:

$$\frac{R_9 : \neg(P_{1,2} \vee P_{2,1})}{R_{10} : \neg P_{1,2} \wedge \neg P_{2,1}}$$

- Done! Notice that here I did not use model checking.

Monotonicity I

- Note that for previous example we want to arrive at $\neg P_{1,2} \land \neg P_{2,1}$
- So we can limit search to sentences relevant to $P_{1,2}$ and $P_{2,1}$ (directly or indirectly)
- It's possible to prove a sentence without using the whole KB: If $KB \models P$, then $KB \land Q \models P$ (**Monotonicity**)

Monotonicity II

- Make sure you understand the previous example. If you make a sequence of inferences starting with $\{S_1, S_2, ..., S_n\}$ and each inference begin with $\{S_1, S_2, ..., S_n\}$ or from what was inferred, and the last sentence inferred is S, then $\{S_1, S_2, ..., S_n\} \models S$.
  For instance suppose you start with $\{A, B, C\}$ and do the following:
  - $\{A, B\}$ infers D
  - $\{B, C\}$ infers E
  - $\{A, D\}$ infers F
  - $\{E, F\}$ infers G

  Then altogether $\{A, B, C\} \models G$.

Monotonicity III

- Exercise. Prove the following: If $S \models S'$ and $S \models S''$, then $S \models S' \wedge S''$.

CNF I

- To get ready for the resolution method for proving entailment, we will need the concept of conjunctive normal form.
- **Literal** = propositional variable with or without $\neg$.
- Example. Let P, Q be propositional variables.
  - P and $\neg P$ are both literals.
  - $P \wedge Q$ and $\neg(P \wedge Q)$ are both not literals.

CNF II

- A sentence is in the **conjunctive normal form (CNF)** if it looks like this:

$$(P \vee \neg R \vee Q \vee S) \wedge (R \vee Q) \wedge (\neg S \vee \neg R \vee P)$$

  i.e., conjunction of disjunction of literals, i.e., "and's of or's (or literals)".
- In the above CNF, the three "or" subsentences (disjunctions)

$$P \vee \neg R \vee Q \vee S, \quad R \vee Q, \quad \neg S \vee \neg R \vee P$$

  are **clauses** of the CNF.
- FACT: Every propositional sentence is equivalent to a CNF.
- k**CNF** $=$ a CNF where every clause has k literals. Example of 3CNF: $(A \vee B \vee \neg C) \wedge (A \vee C \vee D)$.

CNF III

- A sentence is in the **disjunctive normal form (DNF)** if it looks like this:

$$(P \land \neg R \land Q \land S) \lor (R \land Q) \lor (\neg S \land \neg R \land P)$$

i.e., disjunction of conjunction of literals, i.e., "or's of and's (of literals)". So basically the "mirror image" of CNF.

CNF IV

- Algorithm to convert any proposition to CNF:
  1. Remove $\leftrightarrow$ using biconditional elimination
  2. Remove $\rightarrow$ using implication elimination
  3. Move $\neg$ inside (...) using idempotency, de Morgan's rule
  4. Use distributivity

CNF V

- Note that

$$\begin{aligned}
&(A \wedge B \wedge C \wedge D) \vee (E \wedge F) \\
&\equiv (A \vee (E \wedge F)) \wedge (B \vee (E \wedge F)) \wedge \cdots \\
&\equiv ((A \vee E) \wedge (A \vee F)) \wedge ((B \vee E) \wedge (B \vee F)) \wedge \cdots \\
&\equiv (A \vee E) \wedge (A \vee F) \wedge (B \vee E) \wedge (B \vee F) \wedge \cdots
\end{aligned}$$

i.e., “or’s of and’s” can always be converted to “and’s of or’s”, i.e., disjunctive normal forms (DNF) can always be converted to conjunctive normal forms (CNF).

CNF VI

- Example. Rewrite $B_{1,1} \leftrightarrow P_{1,2} \vee P_{2,1}$ in 3-CNF form.
  Solution.

$$\begin{aligned}
& B_{1,1} \leftrightarrow P_{1,2} \vee P_{2,1} \\
& \equiv (B_{1,1} \rightarrow P_{1,2} \vee P_{2,1}) \wedge (P_{1,2} \vee P_{2,1} \rightarrow B_{1,1}) \\
& \equiv (\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1}) \\
& \equiv (\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1}) \\
& \equiv (\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})) \\
& \equiv (\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})
\end{aligned}$$

- The following example gives you a more systematic approach.

CNF VII

- Example. Rewrite $A \wedge B \to \neg(C \vee D \vee E)$ in CNF form.
  Solution.
  Remove biconditionals: Done since there is no biconditionals.

CNF VIII

Remove implications:

$$\begin{aligned} & A \wedge B \rightarrow \neg(C \vee D \wedge E) \\ & \quad \equiv (\neg(A \wedge B) \vee \neg(C \vee D \wedge E)) \end{aligned}$$

CNF IX

Use de Morgan's law and $\neg\neg P \equiv P$ (move $\neg$ as far inward as possible):

$$
\begin{aligned}
& A \wedge B \rightarrow \neg(C \vee D \wedge E) \\
& \quad \equiv (\neg(A \wedge B) \vee \neg(C \vee D \wedge E)) \\
& \quad \equiv (\neg A \vee \neg B) \vee \neg(C \vee D \wedge E)) \\
& \quad \equiv (\neg A \vee \neg B) \vee \neg(C \vee (D \wedge E))) \\
& \quad \equiv (\neg A \vee \neg B) \vee (\neg C \wedge \neg(D \wedge E)) \\
& \quad \equiv (\neg A \vee \neg B) \vee (\neg C \wedge (\neg D \vee \neg E))
\end{aligned}
$$

CNF X

Distribute $\wedge$ into $\vee$ to get DNF:

$$\begin{aligned}
&A \wedge B \rightarrow \neg(C \vee D \wedge E) \\
&\quad \equiv (\neg A \vee \neg B) \vee (\neg C \wedge (\neg D \vee \neg E)) \\
&\quad \equiv (\neg A) \vee (\neg B) \vee (\neg C \wedge (\neg D \vee \neg E)) \\
&\quad \equiv (\neg A) \vee (\neg B) \vee ((\neg C \wedge \neg D) \vee (\neg C \wedge \neg E)) \\
&\quad \equiv (\neg A) \vee (\neg B) \vee (\neg C \wedge \neg D) \vee (\neg C \wedge \neg E)
\end{aligned}$$

Rearrange the clauses so that the clauses with 1 literal appears first (like the above).

CNF XI

Then all the clauses with 1 literals together forms a CNF with 1 clause:

$$\begin{aligned} &A \wedge B \to \neg(C \vee D \wedge E) \\ &\quad \equiv \underline{(\neg A) \vee (\neg B)} \vee (\neg C \wedge \neg D) \vee (\neg C \wedge \neg E) \end{aligned}$$

CNF XII

Distribute the CNF into $(\neg C \wedge \neg D)$:

$$\begin{aligned}
&A \wedge B \rightarrow \neg(C \vee D \wedge E) \\
&\quad \equiv \underline{(\neg A \vee \neg B)} \vee (\neg C \wedge \neg D) \vee (\neg C \wedge \neg E) \\
&\quad \equiv ((\underline{(\neg A \vee \neg B)} \vee \neg C) \wedge (\underline{(\neg A \vee \neg B)} \vee \neg D)) \vee (\neg C \wedge \neg E) \\
&\quad \equiv \underline{((\neg A \vee \neg B \vee \neg C) \wedge (\neg A \vee \neg B \vee \neg D))} \vee (\neg C \wedge \neg E)
\end{aligned}$$

You get a CNF (see underlined).

CNF XIII

Distribute the CNF again to the right:

$$
\begin{aligned}
& A \wedge B \rightarrow \neg(C \vee D \wedge E) \\
& \quad \equiv \underline{((\neg A \vee \neg B \vee \neg C) \wedge (\neg A \vee \neg B \vee \neg D))} \vee (\neg C \wedge \neg E) \\
& \quad \equiv (\underline{((\neg A \vee \neg B \vee \neg C) \wedge (\neg A \vee \neg B \vee \neg D))} \vee \neg C) \\
& \qquad \wedge (\underline{((\neg A \vee \neg B \vee \neg C) \wedge (\neg A \vee \neg B \vee \neg D))} \vee \neg E)
\end{aligned}
$$

CNF XIV

But notice that this is not a CNF:

$$(\underline{((\neg A \vee \neg B \vee \neg C) \wedge (\neg A \vee \neg B \vee \neg D))} \vee \neg C)$$

That's OK: distribute $\neg C$ to the left and you get a 2CNF:

$$\begin{aligned}
&A \wedge B \to \neg(C \vee D \wedge E) \\
&\quad \equiv (\underline{((\neg A \vee \neg B \vee \neg C) \wedge (\neg A \vee \neg B \vee \neg D))} \vee \neg C) \\
&\qquad \wedge (\underline{((\neg A \vee \neg B \vee \neg C) \wedge (\neg A \vee \neg B \vee \neg D))} \vee \neg E) \\
&\quad \equiv (\underline{((\neg A \vee \neg B \vee \neg C \vee \neg C) \wedge (\neg A \vee \neg B \vee \neg D \vee \neg C))}) \\
&\qquad \wedge (\underline{((\neg A \vee \neg B \vee \neg C) \wedge (\neg A \vee \neg B \vee \neg D))} \vee \neg E)
\end{aligned}$$

CNF XV

Also, distribute $\neg E$ to the left:

$$\begin{aligned} & A \land B \to \neg(C \lor D \land E) \\ & \quad \equiv (\underline{((\neg A \lor \neg B \lor \neg C \lor \neg C) \land (\neg A \lor \neg B \lor \neg D \lor \neg C))} \\ & \qquad \land (\underline{((\neg A \lor \neg B \lor \neg C) \land (\neg A \lor \neg B \lor \neg D))} \lor \neg E) \\ & \quad \equiv (\underline{((\neg A \lor \neg B \lor \neg C \lor \neg C) \land (\neg A \lor \neg B \lor \neg D \lor \neg C))} \\ & \qquad \land (\underline{((\neg A \lor \neg B \lor \neg C \lor \neg E) \land (\neg A \lor \neg B \lor \neg D \lor \neg E))}) \end{aligned}$$

And you get a complete CNF.

CNF XVI

And after some cleanup (for instance $\neg C \vee \neg C \equiv \neg C$),

$$\begin{aligned}
&A \wedge B \rightarrow \neg(C \vee D \wedge E) \\
&\quad \equiv (\neg A \vee \neg B \vee \neg C) \wedge (\neg A \vee \neg B \vee \neg C \vee \neg D) \\
&\qquad \wedge (\neg A \vee \neg B \vee \neg C \vee \neg E) \wedge (\neg A \vee \neg B \vee \neg D \vee \neg E)
\end{aligned}$$

$\square$

CNF XVII

In the above example if the term we are distrbuting into is not a conjunction of two literals $(\neg C \wedge \neg D)$ but is a conjunction of *three* literals it would still work:

$$\begin{aligned}
&\underline{((\neg A) \vee (\neg B))} \vee (\neg C \wedge \neg D \wedge Z) \vee \ldots \\
&\equiv (\underline{(\neg A) \vee (\neg B)} \vee \neg C) \\
&\quad \wedge (\underline{(\neg A) \vee (\neg B)} \vee \neg D) \\
&\quad \wedge (\underline{(\neg A) \vee (\neg B)} \vee Z) \vee \ldots
\end{aligned}$$

CNF XVIII

In the above example we started off with a CNF (see underlined) because we have DNF clauses with 1 literal:

$$\underline{((\neg A) \vee (\neg B))} \vee (\neg C \wedge \neg D) \vee \ldots$$

If you start with a DNF where smallest clauses have 2 iterals (instead of 1) it would still work:

$$\begin{aligned} &\underline{((\neg A \wedge X) \vee (\neg B \wedge Y))} \vee (\neg C \wedge \neg D) \vee \ldots \\ &\equiv \underline{(\neg A \vee \neg B) \wedge (\neg A \vee Y) \wedge (X \vee \neg B) \vee (X \vee \neg Y)} \\ &\quad \vee (\neg C \wedge \neg D) \vee \ldots \end{aligned}$$

i.e., two the two clauses and distribute to get the beginning to be a CNF.

CNF XIX

To skip showing the distribution steps, look at this again:

$$\begin{aligned} & A \wedge B \rightarrow \neg(C \vee D \wedge E) \\ & \quad \equiv (\neg A) \vee (\neg B) \vee (\neg C \wedge \neg D) \vee (\neg C \wedge \neg E) \end{aligned}$$

There are 4 clauses. Forms all groups with 4 literals, one from each clause:

$$\begin{aligned} & A \wedge B \rightarrow \neg(C \vee D \wedge E) \\ & \equiv (\neg A) \vee (\neg B) \vee (\neg C \wedge \neg D) \vee (\neg C \wedge \neg E) \\ & \equiv \neg A, B, \neg C, \neg C \qquad \neg A, B, \neg C, \neg E \\ & \qquad \neg A, B, \neg D, \neg C \qquad \neg A, B, \neg D, \neg E \end{aligned}$$

CNF XX

Put in the operators between the clauses:

$$\begin{aligned}
&A \wedge B \to \neg(C \vee D \wedge E) \\
&\equiv (\neg A) \vee (\neg B) \vee (\neg C \wedge \neg D) \vee (\neg C \wedge \neg E) \\
&\equiv \neg A \vee B \vee \neg C \vee \neg C \qquad \neg A \vee B \vee \neg C \vee \neg E \\
&\qquad \neg A \vee B \vee \neg D \vee \neg C \qquad \neg A \vee B \vee \neg D \vee \neg E
\end{aligned}$$

and then this

$$\begin{aligned}
&A \wedge B \to \neg(C \vee D \wedge E) \\
&\equiv (\neg A) \vee (\neg B) \vee (\neg C \wedge \neg D) \vee (\neg C \wedge \neg E) \\
&\equiv (\neg A \vee \neg B \vee \neg C \vee \neg C) \wedge (\neg A \vee \neg B \vee \neg C \vee \neg E) \wedge \\
&\qquad (\neg A \vee \neg B \vee \neg D \vee \neg C) \wedge (\neg A \vee \neg B \vee \neg D \vee \neg E)
\end{aligned}$$

CNF XXI

- Exercise. Rewrite the following in CNF:
  - $\neg(A \vee B \vee C \wedge D)$
  - $A \vee B \rightarrow \neg(C \wedge \neg D)$
  - $((A \vee B) \rightarrow (B \wedge C)) \leftrightarrow (\neg A \vee \neg B \wedge \neg C)$
- Exercise. Can you find an algorithm that produces an equivalent CNF with the least number of clauses? The least number of literals?

Resolution I

- Why did I cover rewriting a proposition in CNF?
- Because it will be used in resolution rules to show that $K \models S$ ...

Resolution II

- THEOREM. $K \wedge \neg S$ is not satisfiable iff $K \models S$.

  Proof.
  ($\implies$):

Resolution III

($\Longleftarrow$):

$\square$

Resolution IV

- THEOREM. A proposition P is not satisfiable iff $P \models \text{FALSE}$.
  Proof.
  $(\implies)$:

  $(\impliedby)$:

  $\square$

Resolution V

- We now have

$$K \models S \iff K \wedge \neg S \text{ not satisfiable } \iff K \wedge \neg S \models \textsc{False}$$

- So to show $K \models S$, we have to show $K \wedge \neg S$ is not satisfiable which is the same as proving $K \wedge \neg S \models \textsc{False}$.
- We will use **resolution rule**:

$$\frac{P \vee Q \vee R \vee T,\ \ A \vee \neg R \vee B \vee C}{P \vee Q \vee T \vee A \vee B \vee C}$$

$P \vee Q \vee T \vee A \vee B \vee C$ is called the **resolvent**. Note that the resolution is based on one pair of **complementary literals**. In the above, the pair is $R, \neg R$.

Resolution VI

- It's possible for two clauses to have more than one pair of complementary literals.
- Important fact: resolution takes two disjunctions and produce a disjunction.
- Therefore we'll write $K \wedge \neg S$ as a CNF and perform resolution again and again until we get $\textsc{False}$.

Resolution VII

- Example: Let K be our KB where $K \equiv (B_{1,1} \leftrightarrow (P_{1,2} \vee P_{2,1})) \wedge \neg B_{1,1}$. Want $K \models \neg P_{1,2}$.
  - If I can show $K \wedge \neg\neg P_{1,2}$ is not satisfiable, then I conclude that $K \models \neg P_{1,2}$.
  - Write K in CNF:

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

  - Write $K \wedge \neg\neg P_{1,2}$ in CNF

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

  - I now use resolution on pairs of clauses of $K \wedge \neg\neg P_{1,2}$ until I infer FALSE.

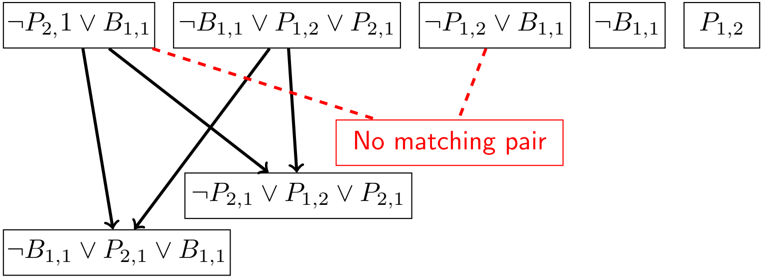
Resolution VIII

$$\begin{tikzcd}
\boxed{\underline{\neg P_{2,1}} \lor B_{1,1}} \arrow[d] & \boxed{\neg B_{1,1} \lor P_{1,2} \lor \underline{P_{2,1}}} \arrow[dl] & \boxed{\neg P_{1,2} \lor B_{1,1}} & \boxed{\neg B_{1,1}} & \boxed{P_{1,2}} \\
\boxed{\neg B_{1,1} \lor P_{2,1} \lor B_{1,1}} & & & &
\end{tikzcd}$$

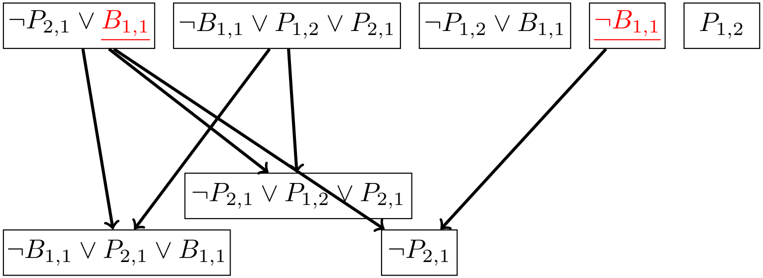
Resolution IX

$$\begin{tikzcd}
\boxed{\neg P_{2,1} \vee \underline{B_{1,1}}} \arrow[dd] \arrow[dr] & \boxed{\underline{\neg B_{1,1}} \vee P_{1,2} \vee P_{2,1}} \arrow[d] \arrow[ddl] & \boxed{\neg P_{1,2} \vee B_{1,1}} & \boxed{\neg B_{1,1}} & \boxed{P_{1,2}} \\
 & \boxed{\neg P_{2,1} \vee P_{1,2} \vee P_{2,1}} & & & \\
\boxed{\neg B_{1,1} \vee P_{2,1} \vee B_{1,1}} & & & &
\end{tikzcd}$$

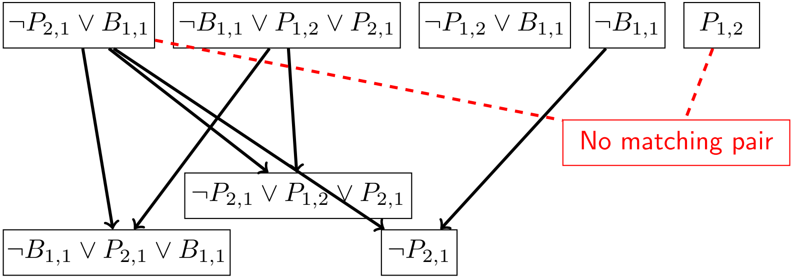
Resolution X


$\neg P_2,1 \lor B_{1,1}$
$\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}$
$\neg P_{1,2} \lor B_{1,1}$
$\neg B_{1,1}$
$P_{1,2}$
No matching pair
$\neg P_{2,1} \lor P_{1,2} \lor P_{2,1}$
$\neg B_{1,1} \lor P_{2,1} \lor B_{1,1}$

Resolution XI


$\neg P_{2,1} \lor \underline{B_{1,1}}$
$\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}$
$\neg P_{1,2} \lor B_{1,1}$
$\underline{\neg B_{1,1}}$
$P_{1,2}$
$\neg P_{2,1} \lor P_{1,2} \lor P_{2,1}$
$\neg B_{1,1} \lor P_{2,1} \lor B_{1,1}$
$\neg P_{2,1}$

Resolution XII


$\neg P_{2,1} \vee B_{1,1}$
$\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}$
$\neg P_{1,2} \vee B_{1,1}$
$\neg B_{1,1}$
$P_{1,2}$
No matching pair
$\neg P_{2,1} \vee P_{1,2} \vee P_{2,1}$
$\neg B_{1,1} \vee P_{2,1} \vee B_{1,1}$
$\neg P_{2,1}$

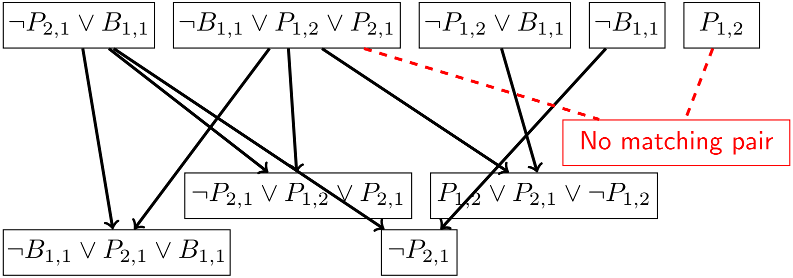
Resolution XIII

$$\begin{tikzcd}
\boxed{\neg P_{2,1} \lor B_{1,1}} \arrow[dd] \arrow[dr] \arrow[ddrr] & \boxed{\underline{\textcolor{red}{\neg B_{1,1}}} \lor P_{1,2} \lor P_{2,1}} \arrow[ddl] \arrow[d] \arrow[dr] & \boxed{\neg P_{1,2} \lor \underline{\textcolor{red}{B_{1,1}}}} \arrow[d] & \boxed{\neg B_{1,1}} \arrow[ddl] & \boxed{P_{1,2}} \\
 & \boxed{\neg P_{2,1} \lor P_{1,2} \lor P_{2,1}} & \boxed{P_{1,2} \lor P_{2,1} \lor \neg P_{1,2}} & & \\
\boxed{\neg B_{1,1} \lor P_{2,1} \lor B_{1,1}} & & \boxed{\neg P_{2,1}} & &
\end{tikzcd}$$

Resolution XIV

$$\begin{tikzcd}
\boxed{\neg P_{2,1} \lor B_{1,1}} \arrow[dd] \arrow[dr] \arrow[ddrr] & \boxed{\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}} \arrow[ddl] \arrow[d] \arrow[dr] \arrow[drrr, dashed, no head] & \boxed{\neg P_{1,2} \lor B_{1,1}} \arrow[d] & \boxed{\neg B_{1,1}} \arrow[ddl] \arrow[dr, dashed, no head] & \boxed{P_{1,2}} \\
 & \boxed{\neg P_{2,1} \lor P_{1,2} \lor P_{2,1}} & \boxed{P_{1,2} \lor P_{2,1} \lor \neg P_{1,2}} & & \boxed{\text{No matching pair}} \\
\boxed{\neg B_{1,1} \lor P_{2,1} \lor B_{1,1}} & & \boxed{\neg P_{2,1}} & &
\end{tikzcd}$$

Resolution XV


$\neg P_{2,1} \lor B_{1,1}$
$\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}$
$\neg P_{1,2} \lor B_{1,1}$
$\neg B_{1,1}$
$P_{1,2}$
No matching pair
$\neg P_{2,1} \lor P_{1,2} \lor P_{2,1}$
$P_{1,2} \lor P_{2,1} \lor \neg P_{1,2}$
$\neg B_{1,1} \lor P_{2,1} \lor B_{1,1}$
$\neg P_{2,1}$

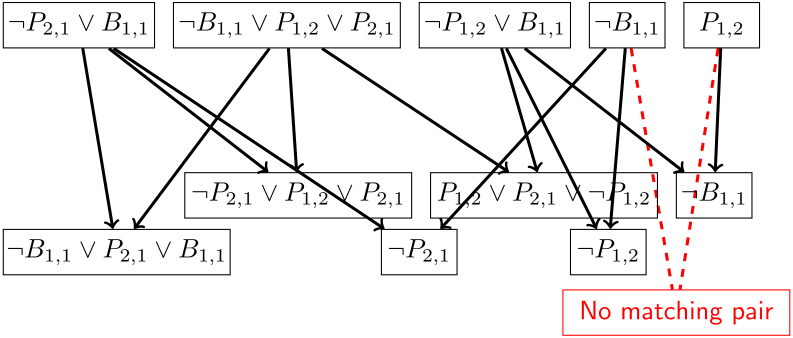
Resolution XVI

$$\begin{tikzcd}
\boxed{\neg P_{2,1} \lor B_{1,1}} \arrow[dd] \arrow[drr] \arrow[ddrrr] & \boxed{\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}} \arrow[ddl] \arrow[dr] \arrow[drrr] & & \boxed{\neg P_{1,2} \lor \underline{B_{1,1}}} \arrow[dr] \arrow[ddr] & \boxed{\underline{\neg B_{1,1}}} \arrow[ddl] \arrow[dd] & \boxed{P_{1,2}} \\
& & \boxed{\neg P_{2,1} \lor P_{1,2} \lor P_{2,1}} & & \boxed{P_{1,2} \lor P_{2,1} \lor \neg P_{1,2}} & \\
\boxed{\neg B_{1,1} \lor P_{2,1} \lor B_{1,1}} & & & \boxed{\neg P_{2,1}} & \boxed{\neg P_{1,2}} &
\end{tikzcd}$$

Resolution XVII

$$\begin{tikzcd}
\boxed{\neg P_{2,1} \lor B_{1,1}} \arrow[dd] \arrow[drr] \arrow[ddrrr] & & \boxed{\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}} \arrow[ddll] \arrow[d] \arrow[drr] & & \boxed{\underline{\neg P_{1,2}} \lor B_{1,1}} \arrow[d] \arrow[ddr] \arrow[drr] & \boxed{\neg B_{1,1}} \arrow[ddll] \arrow[dd] & \boxed{\underline{P_{1,2}}} \arrow[d] \\
& & \boxed{\neg P_{2,1} \lor P_{1,2} \lor P_{2,1}} & & \boxed{P_{1,2} \lor P_{2,1} \lor \neg P_{1,2}} & & \boxed{\neg B_{1,1}} \\
\boxed{\neg B_{1,1} \lor P_{2,1} \lor B_{1,1}} & & & \boxed{\neg P_{2,1}} & & \boxed{\neg P_{1,2}} &
\end{tikzcd}$$

Resolution XVIII


$\neg P_{2,1} \lor B_{1,1}$
$\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}$
$\neg P_{1,2} \lor B_{1,1}$
$\neg B_{1,1}$
$P_{1,2}$
$\neg P_{2,1} \lor P_{1,2} \lor P_{2,1}$
$P_{1,2} \lor P_{2,1} \lor \neg P_{1,2}$
$\neg B_{1,1}$
$\neg B_{1,1} \lor P_{2,1} \lor B_{1,1}$
$\neg P_{2,1}$
$\neg P_{1,2}$
No matching pair

Resolution XIX

$$\begin{tikzcd}
\boxed{\neg P_{2,1} \lor B_{1,1}} \arrow[dd] \arrow[dr] \arrow[ddrr] & \boxed{\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}} \arrow[ddl] \arrow[d] \arrow[drr] & & \boxed{\neg P_{1,2} \lor B_{1,1}} \arrow[d] \arrow[ddr] \arrow[drr] & \boxed{\neg B_{1,1}} \arrow[ddll] \arrow[dd] & \boxed{P_{1,2}} \arrow[d] \arrow[ddd] \\
& \boxed{\neg P_{2,1} \lor P_{1,2} \lor P_{2,1}} & & \boxed{P_{1,2} \lor P_{2,1} \lor \neg P_{1,2}} & & \boxed{\neg B_{1,1}} \\
\boxed{\neg B_{1,1} \lor P_{2,1} \lor B_{1,1}} & & \boxed{\neg P_{2,1}} & & \boxed{\neg P_{1,2}} \arrow[dr] & \\
& & & & & \boxed{}
\end{tikzcd}$$

… etc. …

Resolution XX

$$\begin{tikzcd}
\boxed{\neg P_{2,1} \lor B_{1,1}} \arrow[dd] \arrow[dr] \arrow[ddrr] & \boxed{\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}} \arrow[ddl] \arrow[d] \arrow[drr] & & \boxed{\neg P_{1,2} \lor B_{1,1}} \arrow[d] \arrow[ddr] \arrow[drr] & \boxed{\neg B_{1,1}} \arrow[ddll] \arrow[dd] & \boxed{P_{1,2}} \arrow[d] \arrow[ddd] \\
& \boxed{\neg P_{2,1} \lor P_{1,2} \lor P_{2,1}} & & \boxed{P_{1,2} \lor P_{2,1} \lor \neg P_{1,2}} & & \boxed{\neg B_{1,1}} \\
\boxed{\neg B_{1,1} \lor P_{2,1} \lor B_{1,1}} & & \boxed{\neg P_{2,1}} & & \boxed{\neg P_{1,2}} \arrow[dr] & \\
\dots \text{ etc. } \dots & & & \boxed{\text{Empty clause. } \textsc{False}\text{. So } K \models \neg P_{1,2}.} \arrow[rr, red] & & \boxed{}
\end{tikzcd}$$

Resolution XXI

- Given two clauses $C1, C2$, let `PL_RESOLVE(C1, C2)` be the list of all unique resolvents of $C1, C2$. We return a list of resolvents because there might be more than one pair of complementary literals (but see below).
- Example. `PL_RESOLVE`$(P \vee Q \vee R, \neg R \vee B \vee C) = [P \vee Q \vee B \vee C]$ finds a pair of complementary literals in the two clauses and then create a disjunction of all literals except for that pair and return a list of that disjunction.
- Implementation-wise, it should be clear from the above example that a clause will be implemented as a list of literals. For instance $A \vee \neg B$ is modeled as $[A, \neg B]$,

Resolution XXII

- In that case, for the binary resolution law, mathematically speaking it is

$$\frac{P \vee Q, \quad \neg Q \vee R}{\therefore P \vee R}$$

In implementation it becomes

$$\texttt{PL\_RESOLVE}([P, Q], [\neg Q, R]) = [[P, R]]$$

i.e., $[P, R]$ is just a list of literals in $[P, Q]$ and $[\neg Q, R]$ except for the complementary pair of literals.

Resolution XXIII

- If we do that, then

$$\texttt{PL\_RESOLVE}([R], [\neg R]) = [[\ ]]$$

  Mathematically the result should be $\textsc{False}$. Therefore we need to remember that empty list $[\ ]$ as a disjunction means $\textsc{False}$.

Resolution XXIV

- Example. Case of at least two complementary literals. $C1 = A \vee B \vee C$ and $C2 = \neg A \vee \neg B \vee C$, then $\texttt{PL\_RESOLVE}(C1, C2)$ returns
  - Pair $A, \neg A$: Get $B \vee C \vee \neg B \vee C = B \vee \neg B \vee C \vee C = \textsc{True}$
  - Pair $B, \neg B$: Get $A \vee C \vee \neg A \vee C = A \vee \neg A \vee C \vee C = \textsc{True}$

  i.e., $\texttt{PL\_RESOLVE}(C1, C2) = [\textsc{True}]$.
- The above example shows that if two clauses $C1, C2$ have at least two pairs of complementary literals, then $\texttt{PL\_RESOLVE}(C1, C2) = [\textsc{True}]$.

Resolution XXV

- However note that we will be adding resolvents to ourknowledge base to prove entailment. Adding TRUE to a knowledge base does not change the knowledge. Therefore we modify the definition of `PL_RESOLVE`$(C1, C2)$. It returns the unique resolvents of $C1, C2$ except for TRUE.

Resolution XXVI

- Hence given two clauses C1, C2 which are disjunctions, representing them as lists of literals,
  - If $C1, C2$ has exactly one pair of complementary literals:

    $$\texttt{PL\_RESOLVE(C1, C2)} = [C]$$

    where C is the list of unique literals from $C1, C2$ except for the complementary pair.
  - Otherwise

    $$\texttt{PL\_RESOLVE(C1, C2)} = [\ ]$$

Resolution XXVII

- Main idea: Let K be a KB and S a sentence. We want to show $K \models S$.
    - Write $K \wedge \neg S$ in CNF form and put clauses into a set `Cs` where each clause is represented as a list of literals.
    - Take a pair of clauses from the `Cs` with exactly one pair of complementary literals and use `PL_RESOLVE` produce a list of a new clause `[C]` and put `C` into the `Cs` if it's not already in there
    - Repeat the above until:
        1. If `PL_RESOLVE` returns $[[\,]]$, conclude that $\underline{K \models S}$.
        2. Otherwise when no new clauses can be added to `Cs`, conclude that $\underline{K \text{ does not} \models S}$.

Resolution

```
ALGORITHM:  PL_RESOLUTION
INPUTS: K - knowledge base as a propositional sentence
        S - propositional sentence
OUTPUT: TRUE iff K entails S

Cs = set of clauses from K ∧ S
new = empty set for new clauses created during resolution

while True:
    for each pair of distinct clauses C1,C2 in Cs:
        resolvents = PL_RESOLVE(C1, C2)
        if resolves contains empty clause [], return TRUE
        new = new ∪ resolvents
    if all clauses in new are already in Cs, return FALSE
    clauses = clauses ∪ new
```

Resolution I

- This is a very simple algorithm
- Exercise. There is a simple way to improve the algorithm. Clauses are re-processed. Improve it. □
- If S is a set of clauses, then the **resolution closure of S**, $RC(S)$, is just the set of clauses you get after repeated application of resolution law until the process stops.
    - The process stops because given a finite set of n propositional variables, you have $2n$ literals, and the clauses are made up of only these literals. Each clause is essential a list of literals connected by disjunction. Therefore there is at most 2^{2n} clauses.

Resolution II

- THEOREM (GROUND RESOLUTION THEOREM). If a set of clauses is unsatisfiable, then the resolution closure of those clauses contains the empty clause.

Horn clauses I

- There are faster entailment algorithms if KB is of special form.
- **Definite clause** = a disjunction with exactly one literal not negated
  - Example: $A \vee \neg B \vee \neg C$. So?

$$A \vee \neg B \vee \neg C \equiv A \vee \neg(B \wedge C) \equiv (B \wedge C) \rightarrow A$$

- **Fact** = a disjunction with 0 literal not negated
  - Example: $\neg B \vee \neg C$. So?

$$\neg B \vee \neg C \equiv \textsc{False} \vee \neg B \vee \neg C \equiv (B \wedge C) \rightarrow \textsc{False}$$

- So definite clauses and facts are equivalent to implications where left-hand side is conjunction and right-hand side is an un-negated variable or $\textsc{False}$.

Horn clauses II

- **Horn clause** = disjunction with at most one (exactly 0 or 1) literal is not negated = definite clause or fact.
  - $(A \vee \neg B \vee \neg C)$ is a Horn clause
  - $(\neg B \vee \neg C)$ is a Horn clause
  - $(A \vee \neg B \vee C)$ is not a Horn clause

Forward and backward chaining I

- Assume KB is made up of Horn clauses.
- "Problem" with resolution method: You do not know where you're heading
- Forward and backward chaining are inference algorithms that take linear time
    - Forward chaining is a form of data-driven reasoning
    - Backward chaining is a form of goal-directed reasoning

Forward and backward chaining II

- Example. Suppose K is made up of

$$\begin{aligned} &P \to Q \\ &L \wedge M \to P \\ &B \wedge L \to M \\ &A \wedge P \to L \\ &A \wedge B \to L \\ &A \\ &B \end{aligned}$$

Show that $K \models Q$.

Forward and backward chaining III

K :

$P \to Q$

$L \wedge M \to P$

$B \wedge L \to M$

$A \wedge P \to L$

$A \wedge B \to L$

A

B

`Aim:` $K \models Q$

Forward and backward chaining IV

$K:$

$P \to Q$

$L \wedge M \to P$

$B \wedge L \to M$

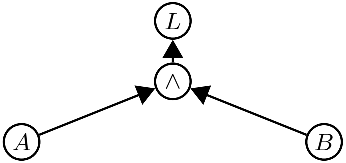
$A \wedge P \to L$

$A \wedge B \to L$

A

B

Aim: $K \models Q$



Forward and backward chaining V

K :

$P \to Q$

$L \wedge M \to P$

$B \wedge L \to M$

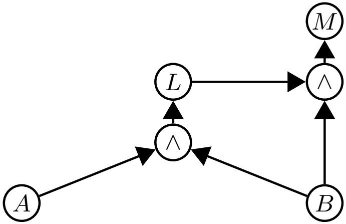
$A \wedge P \to L$

$A \wedge B \to L$

A

B

Aim: $K \models Q$



Forward and backward chaining VI

K :

$P \to Q$

$L \wedge M \to P$

$B \wedge L \to M$

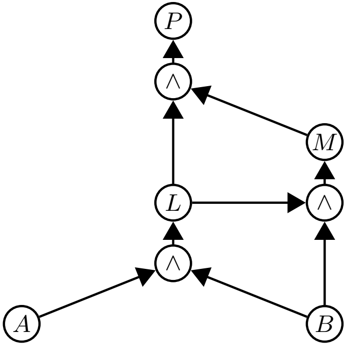
$A \wedge P \to L$

$A \wedge B \to L$

A

B

Aim: $K \models Q$



Forward and backward chaining VII

K :

$P \to Q$

$L \wedge M \to P$

$B \wedge L \to M$

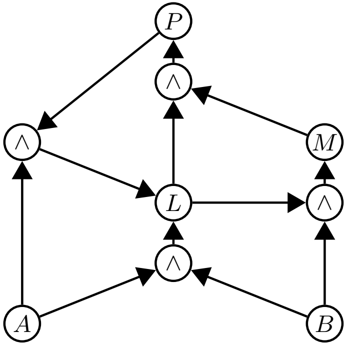
$A \wedge P \to L$

$A \wedge B \to L$

A

B

Aim: $K \models Q$



Forward and backward chaining VIII

K :

$P \to Q$

$L \land M \to P$

$B \land L \to M$

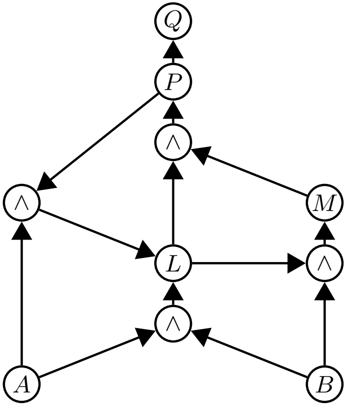
$A \land P \to L$

$A \land B \to L$

A

B

`Aim:` $K \models Q$



Forward and backward chaining IX

- Backward chaining: idea same as forward chaining but start with query Q and work backwards
- Time is also linear

Hard satisfiability problems I

- It's harder to satisfy a CNF when number of propositional variables is small and number of clauses large. (Why?)
- Let m = number of clauses, n = no. of symbols
- Probablity of satisfiability of randomly generated clauses drop rapidly when $m/n = 4.3$
- Apparently:
    - DP and WALKSAT need more time near $m/n = 4.3$
    - DP is quite effective
    - WALKSAT is faster than DP

Davis-Putnam algorithm I

- Recall `TT_ENTAILS` involves exhaustive model checking
- Here are some improvements. Assume propositional sentence is in CNF.
- **Early termination**:
  - $(\ldots \vee \textsc{True} \vee \ldots)$ is $\textsc{True}$. If a DNF has a $\textsc{True}$ clause, then the DNF is $\textsc{True}$.
  - $(\ldots) \wedge \textsc{False} \wedge (\ldots)$ is $\textsc{False}$. If a CNF has a $\textsc{False}$ clause, then the CNF is $\textsc{False}$.
- **Pure symbol heuristics**:
  - A symbol B is **pure** either only B appears or only $\neg B$ appears, but not both
  - If B is pure and B appears, assign true to B
  - If B is pure and $\neg B$ appears, assign false to B

Davis-Putnam algorithm II

- **Unit clause heuristics**:
  - A **unit clause** is a clause with one literal
  - Assign TRUE to literal in unit clauses
- When `TT_ENTAILS` is modified to include the above ideas, you get a fast satisfiability alg.

Davis-Putnam algorithm III

- Example: Is $(A \lor B \lor \neg C) \land (A \lor \neg C) \land (\neg C) \land (\neg B \lor C) \land (A \lor C)$ satisfiable?
  - $\neg C$ is a unit clause. Set $\neg C$ to $\textsc{True}$, i.e., set C to $\textsc{False}$.
  - We get $(A \lor B \lor \textsc{True}) \land (A \lor \textsc{True}) \land (\textsc{True}) \land (\neg B \lor \textsc{False}) \land (A \lor \textsc{False})$
  - By early termination $(\textsc{True}) \land (\textsc{True}) \land (\textsc{True}) \land (\neg B) \land (A)$ which is $(\neg B) \land (A)$.
  - By unit clause heuristics, set B to $\textsc{False}$ and A to $\textsc{True}$.

So the sentence is satisfiable. □

WALKSAT algorithm I

- **WALKSAT** = Local search technique for solving SAT problem. Similar to hill climing
- Input is a proposition as a list of clauses.
- Goal is to find a model (if clauses are satisfiable).
- Evaluation function on an assignment is number of unsatisfied clauses. Therefore we want to minimize evaluation function.
- Initialize by making random assignment A for every propositional variable in clauses. Continually A in a loop, flipping the boolean value assigned to one propositional variable in each iteration.

WALKSAT algorithm II

- The following describes an interation of the loop:
  (1) Randomly pick an unsatisfied clause C
  (2) Roll a die and randomly decide to do (2a) or (2b):
    (2a) Either: pick a symbol P in C and flip the value assigned to P in the assignment
    (2b) Or: go through all the variables in C and flip the value assigned in the assignment. Keep the one that gives minimum number of unsatisfied clauses
- Return the assignment (as a model) if all clauses are satisfied.
- Note:
  - At (2), you can set a probability p to choose (2a). For instance can set p to 0.5.

WALKSAT algorithm III

  - At (2b), you will need a way to break ties, for instance pick randomly among the best.
- If the clauses are not satisfiable, this can run forever. So have a counter and let counter run up to a maximum number of steps.
- If counter reached max number of steps without finding a model, return FAILURE.
- If algorithm returns FAILURE, it can mean either the clauses are not satisfiable or the clauses are satisfiable but not enough time was given.
- In max number of steps is set to ∞, algorithm will run forever if clauses are not satisfiable.

WALKSAT algorithm IV

- WALKSAT is an example of a **Las Vegas** randomized algorithm.

WALKSAT algorithm V

- Example. Check if $C_1 \wedge C_2 \wedge C_3$ is satisfiable where
  $C_1 = W \vee \neg X, \;\; C_2 = X \vee Y \vee Z, \;\; C_3 = W \vee Y \vee \neg Z$
  $Cs = [C_1, C_2, C_3]$ (clauses)
  $p = 0.4$
  $A = \{W = \textsc{False}, X = \textsc{True}, Y = \textsc{False}, Z = \textsc{True}\}$
  - Is A a model for clauses?

WALKSAT algorithm VI

- Example. Check if $C_1 \wedge C_2 \wedge C_3$ is satisfiable where
  $$C_1 = W \vee \neg X, \;\; C_2 = X \vee Y \vee Z, \;\; C_3 = W \vee Y \vee \neg Z$$
  $Cs = [C_1, C_2, C_3]$ (clauses)
  $$p = 0.4$$
  $$A = \{W = \textsc{False}, X = \textsc{True}, Y = \textsc{False}, Z = \textsc{True}\}$$
  - Is A a model for clauses? NO. C_1 not satisfied.

WALKSAT algorithm VII

- Example. Check if $C_1 \wedge C_2 \wedge C_3$ is satisfiable where

  $C_1 = W \vee \neg X, \;\; C_2 = X \vee Y \vee Z, \;\; C_3 = W \vee Y \vee \neg Z$

  $Cs = [C_1, C_2, C_3]$ (clauses)

  $p = 0.4$

  $A = \{W = \text{FALSE}, X = \text{TRUE}, Y = \text{FALSE}, Z = \text{TRUE}\}$

  - Is A a model for clauses? NO. C_1 not satisfied.
  - Iteration 1: Suppose $C = C_1 : W \vee \neg X$ was chosen.

WALKSAT algorithm VIII

- Example. Check if $C_1 \wedge C_2 \wedge C_3$ is satisfiable where

  $C_1 = W \vee \neg X, \;\; C_2 = X \vee Y \vee Z, \;\; C_3 = W \vee Y \vee \neg Z$

  $Cs = [C_1, C_2, C_3]$ (clauses)

  $p = 0.4$

  $A = \{W = \textsc{False}, X = \textsc{True}, Y = \textsc{False}, Z = \textsc{True}\}$

  - Is A a model for clauses? NO. C_1 not satisfied.
  - Iteration 1: Suppose $C = C_1 : W \vee \neg X$ was chosen.
  - Say a rand float 0.25 was generated. $0.25 \leq p$. Execute (2a).

WALKSAT algorithm IX

- Example. Check if $C_1 \wedge C_2 \wedge C_3$ is satisfiable where

  $$C_1 = W \vee \neg X, \;\; C_2 = X \vee Y \vee Z, \;\; C_3 = W \vee Y \vee \neg Z$$

  $Cs = [C_1, C_2, C_3]$ (clauses)

  $p = 0.4$

  $A = \{W = \text{FALSE}, X = \text{TRUE}, Y = \text{FALSE}, Z = \text{TRUE}\}$

  - Is A a model for clauses? NO. C_1 not satisfied.
  - Iteration 1: Suppose $C = C_1 : W \vee \neg X$ was chosen.
  - Say a rand float 0.25 was generated. $0.25 \leq p$. Execute (2a).
  - Suppose X of C_1 was chosen.

WALKSAT algorithm X

- Example. Check if $C_1 \wedge C_2 \wedge C_3$ is satisfiable where

  $C_1 = W \vee \neg X, \;\; C_2 = X \vee Y \vee Z, \;\; C_3 = W \vee Y \vee \neg Z$

  $Cs = [C_1, C_2, C_3]$ (clauses)

  $p = 0.4$

  $A = \{W = \text{FALSE}, X = \text{TRUE}, Y = \text{FALSE}, Z = \text{TRUE}\}$

  - Is A a model for clauses? NO. C_1 not satisfied.
  - Iteration 1: Suppose $C = C_1 : W \vee \neg X$ was chosen.
  - Say a rand float 0.25 was generated. $0.25 \leq p$. Execute (2a).
  - Suppose X of C_1 was chosen.
  - Set A to $\{W = \qquad , X = \qquad , Y = \qquad , Z = \qquad \}$

WALKSAT algorithm XI

- Example. Check if $C_1 \wedge C_2 \wedge C_3$ is satisfiable where
  $$C_1 = W \vee \neg X, \;\; C_2 = X \vee Y \vee Z, \;\; C_3 = W \vee Y \vee \neg Z$$
  $$Cs = [C_1, C_2, C_3] \text{ (clauses)}$$
  $$p = 0.4$$
  $$A = \{W = \textsc{False}, X = \textsc{True}, Y = \textsc{False}, Z = \textsc{True}\}$$
  - Is A a model for clauses? NO. C_1 not satisfied.
  - Iteration 1: Suppose $C = C_1 : W \vee \neg X$ was chosen.
  - Say a rand float 0.25 was generated. $0.25 \leq p$. Execute (2a).
  - Suppose X of C_1 was chosen.
  - Set A to $\{W = \textsc{False}, X = \underline{\textsc{False}}, Y = \textsc{False}, Z = \textsc{True}\}$

WALKSAT algorithm XII

- Example. Check if $C_1 \wedge C_2 \wedge C_3$ is satisfiable where

  $C_1 = W \vee \neg X, \;\; C_2 = X \vee Y \vee Z, \;\; C_3 = W \vee Y \vee \neg Z$

  $Cs = [C_1, C_2, C_3]$ (clauses)

  $p = 0.4$

  $A = \{W = \text{FALSE}, X = \text{FALSE}, Y = \text{FALSE}, Z = \text{TRUE}\}$

  - Is A a model for clauses?

WALKSAT algorithm XIII

- Example. Check if $C_1 \wedge C_2 \wedge C_3$ is satisfiable where

  $C_1 = W \vee \neg X, \;\; C_2 = X \vee Y \vee Z, \;\; C_3 = W \vee Y \vee \neg Z$

  $Cs = [C_1, C_2, C_3]$ (clauses)

  $p = 0.4$

  $A = \{W = \textsc{False}, X = \textsc{False}, Y = \textsc{False}, Z = \textsc{True}\}$

  - Is A a model for clauses? NO. C_3 not satisfied.

WALKSAT algorithm XIV

- Example. Check if $C_1 \wedge C_2 \wedge C_3$ is satisfiable where

$$C_1 = W \vee \neg X, \;\; C_2 = X \vee Y \vee Z, \;\; C_3 = W \vee Y \vee \neg Z$$

$$Cs = [C_1, C_2, C_3] \text{ (clauses)}$$

$$p = 0.4$$

$$A = \{W = \textsc{False}, X = \textsc{False}, Y = \textsc{False}, Z = \textsc{True}\}$$

  - Is A a model for clauses? NO. C_3 not satisfied.
  - Iteration 2: Suppose $C = C_2 : W \vee \neg X$ was chosen.

WALKSAT algorithm XV

- Example. Check if $C_1 \wedge C_2 \wedge C_3$ is satisfiable where

  $C_1 = W \vee \neg X, \;\; C_2 = X \vee Y \vee Z, \;\; C_3 = W \vee Y \vee \neg Z$

  $Cs = [C_1, C_2, C_3]$ (clauses)

  $p = 0.4$

  $A = \{W = \textsc{False}, X = \textsc{False}, Y = \textsc{False}, Z = \textsc{True}\}$

  - Is A a model for clauses? NO. C_3 not satisfied.
  - Iteration 2: Suppose $C = C_2 : W \vee \neg X$ was chosen.
  - Say a rand float 0.9 was generated. $0.9 > p$. Execute (2b).

WALKSAT algorithm XVI

- Example. Check if $C_1 \wedge C_2 \wedge C_3$ is satisfiable where

$$C_1 = W \vee \neg X, \;\; C_2 = X \vee Y \vee Z, \;\; C_3 = W \vee Y \vee \neg Z$$

$$Cs = [C_1, C_2, C_3] \text{ (clauses)}$$

$$p = 0.4$$

$$A = \{W = \textsc{False}, X = \textsc{False}, Y = \textsc{False}, Z = \textsc{True}\}$$

  - Is A a model for clauses? NO. C_3 not satisfied.
  - Iteration 2: Suppose $C = C_2 : W \vee \neg X$ was chosen.
  - Say a rand float 0.9 was generated. $0.9 > p$. Execute (2b).
  - Flip W: num of unsatisfied clauses $=$

WALKSAT algorithm XVII

- Example. Check if $C_1 \wedge C_2 \wedge C_3$ is satisfiable where

$$C_1 = W \vee \neg X, \;\; C_2 = X \vee Y \vee Z, \;\; C_3 = W \vee Y \vee \neg Z$$

$$Cs = [C_1, C_2, C_3] \text{ (clauses)}$$

$$p = 0.4$$

$$A = \{W = \textsc{False}, X = \textsc{False}, Y = \textsc{False}, Z = \textsc{True}\}$$

  - Is A a model for clauses? NO. C_3 not satisfied.
  - Iteration 2: Suppose $C = C_2 : W \vee \neg X$ was chosen.
  - Say a rand float 0.9 was generated. $0.9 > p$. Execute (2b).
  - Flip W: num of unsatisfied clauses = 0

WALKSAT algorithm XVIII

- Example. Check if $C_1 \wedge C_2 \wedge C_3$ is satisfiable where

  $$C_1 = W \vee \neg X, \;\; C_2 = X \vee Y \vee Z, \;\; C_3 = W \vee Y \vee \neg Z$$

  $Cs = [C_1, C_2, C_3]$ (clauses)

  $p = 0.4$

  $A = \{W = \textsc{False}, X = \textsc{False}, Y = \textsc{False}, Z = \textsc{True}\}$

  - Is A a model for clauses? NO. C_3 not satisfied.
  - Iteration 2: Suppose $C = C_2 : W \vee \neg X$ was chosen.
  - Say a rand float 0.9 was generated. $0.9 > p$. Execute (2b).
  - Flip W: num of unsatisfied clauses = 0
  - Flip X: num of unsatisfied clauses =

WALKSAT algorithm XIX

- Example. Check if $C_1 \wedge C_2 \wedge C_3$ is satisfiable where

  $$C_1 = W \vee \neg X, \;\; C_2 = X \vee Y \vee Z, \;\; C_3 = W \vee Y \vee \neg Z$$

  $Cs = [C_1, C_2, C_3]$ (clauses)

  $$p = 0.4$$

  $$A = \{W = \textsc{False}, X = \textsc{False}, Y = \textsc{False}, Z = \textsc{True}\}$$

  - Is A a model for clauses? NO. C_3 not satisfied.
  - Iteration 2: Suppose $C = C_2 : W \vee \neg X$ was chosen.
  - Say a rand float 0.9 was generated. $0.9 > p$. Execute (2b).
  - Flip W: num of unsatisfied clauses = 0
  - Flip X: num of unsatisfied clauses = 2

WALKSAT algorithm XX

- Example. Check if $C_1 \wedge C_2 \wedge C_3$ is satisfiable where

  $$C_1 = W \vee \neg X, \;\; C_2 = X \vee Y \vee Z, \;\; C_3 = W \vee Y \vee \neg Z$$

  $$Cs = [C_1, C_2, C_3] \text{ (clauses)}$$

  $$p = 0.4$$

  $$A = \{W = \text{FALSE}, X = \text{FALSE}, Y = \text{FALSE}, Z = \text{TRUE}\}$$

  - Is A a model for clauses? NO. C_3 not satisfied.
  - Iteration 2: Suppose $C = C_2 : W \vee \neg X$ was chosen.
  - Say a rand float 0.9 was generated. $0.9 > p$. Execute (2b).
  - Flip W: num of unsatisfied clauses = 0
  - Flip X: num of unsatisfied clauses = 2
  - Flip Y: num of unsatisfied clauses = 0
  - Flip Z: num of unsatisfied clauses = 2
  - Set A to $\{W = \underline{\text{TRUE}}, X = \text{FALSE}, Y = \text{FALSE}, Z = \text{TRUE}\}$

WALKSAT algorithm XXI

- Example. Check if $C_1 \wedge C_2 \wedge C_3$ is satisfiable where

  $$C_1 = W \vee \neg X, \;\; C_2 = X \vee Y \vee Z, \;\; C_3 = W \vee Y \vee \neg Z$$

  $$Cs = [C_1, C_2, C_3] \text{ (clauses)}$$

  $$p = 0.4$$

  $$A = \{W = \underline{\textsc{True}}, X = \textsc{False}, Y = \textsc{False}, Z = \textsc{True}\}$$

  - Is A a model for clauses? YES. Return A.

$\square$

- In the above I pick first best when breaking ties. Note that when evaluation function $= 0$ it means I reach a model and can end earlier.

WALKSAT

```
ALGORITHM:  WALKSAT
INPUTS: Cs - set of clauses
        p - probability of choosing step (2a)
        max_flips - max number of flips executed
OUTPUT: satisfying model or FAILURE

let A = random assignment of TRUE/FALSE to variables in Cs
for step = 1, 2, 3, ..., max_flips:
    if clauses(model) is TRUE, return model
    C = randomly chosen clause from Cs that is not
        satisfied by A
    if rand float in [0.0, 1.0] is <= p:
        randomly pick variable in C and flip it's value in A
    else:
        Go through all possible single flip in A and keep the
        one that gives min number of not satisfiable clauses.
        Break ties by picking from best randomly.

return FAILURE
```

WALKSAT algorithm I

- For all the WALKSAT computational exercises below, assume the following:
  - $p = 0.6$
  - max number of steps $= 5$
  - Your random float generator will produce $0.1, 0.9, 0.2, 0.8, 0.3, 0.7, 0.4, 0.6, 0.5$ in that order. (You will not need all of them.)
  - For (2a), assume you pick variable within a clause according to "first variable", "second variable", "first variable", "first variable", "second variable", "first variable" in that order.
  - For (2b), break ties by choosing the first flip.

WALKSAT algorithm II

- Exercise. Let $C_1 = \neg A \vee B \vee C$, $C_2 = A \vee \neg B \vee C$, $C_3 = A \vee B \vee \neg C$. Run WALKSAT on $P = C_1 \wedge C_2 \wedge C_3$. and with an initial assignment of $\{A = \textsc{True}, B = \textsc{False}, C = \textsc{False}\}$. Assume the clause randomly picked is C_1, C_3, C_2 in that order. □
- Exercise. Let $C_1 = A \vee B \vee C$, $C_2 = \neg A \vee B \vee C$, $C_3 = A \vee \neg B \vee C$, $C_4 = A \vee B \vee \neg C$, $C_5 = \neg A \vee \neg B$, Run WALKSAT on $C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5$ and with an initial assignment of $\{A = \textsc{True}, B = \textsc{True}, C = \textsc{True}\}$. Assume the clause randomly picked is $C_1, C_5, C_2, C_5, C_3, C_5$ in that order. □

WALKSAT algorithm III

- Exercise. John suggested the following variation of WALKSAT and wanted to called it JOHN-WALKSAT. You have to iterate over all clauses $C_1, C_2, \ldots$ one at a time to check for satisfiability anyway. So once an unsatisfiable clause C_i clause is found, we should let that be the clause C, instead of randomly picking C. Furthermore, once C_i is found, since we know it's not satisfiable, all we need to do is to set the literal of C_i to be $\textsc{True}$. For instance if $C_i = \neg X \vee Y \neg Z$ is not satisfiable, then $\neg X$, Y, $\neg Z$ must be all set fo $\textsc{False}$ by the assignment. So just let $\neg X$ have the value $\textsc{True}$, i.e., let $X = \textsc{False}$ in assignment A, i.e., just flip X in A. What do you think? □

Knowledge-based agent I

- A knowledge-based agent will have a logic system that includes managing a knowledge base, say K, with TELL and ASK.
- Frequently time is involved. We will assume timestamp is an integer value from values $0, 1, 2, 3, \ldots$, starting at 0.
- Recall that in the tic-tac-toe example earlier, I mentioned that
  - Some propositions are always true
  - Some propositions are true because of observation
  - Some propositions are true because of the agent's action

Knowledge-based agent II

- `TELL(K, MAKE-PERCEPT-SENTENCE(percept, t))` will add a propositional sentence to KB K. The `MAKE-PERCEPT-SENTENCE(percept, t)` will create the appropriate propositional sentence based on what is perceived and together with a timestamp `t`.
- `action = ASK(K, MAKE-ACTION-QUERY(t))` will ask the logic system to provide an action (that maximizing overall performance) current time `t`.

Knowledge-based agent III

- `TELL(K, MAKE-ACTION-SENTENCE(action, t))` will add a propositional sentence to K. The `MAKE-ACTION-SENTENCE(action, t)` will create the appropriate propositional sentence based on the action recommended by the `ASK` command.

Model checking

```
ALGORITHM: KB-AGENT(percept)
INPUT:      percept - a percept (from sensors)
OUTPUT:     action  - an action that will maximize performance
PERSISTENT: KB      - knowledge base
            t       - counter indicating time (initially 0)

xTELL(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
```

Wumpus and propositional logic I

- Previously:
  - $P_{i,j}$ = "Pit at (i, j)"
  - $B_{i,j}$ = "Breeze at (i, j)"
  - $S_{i,j}$ = "Stench at (i, j)"
  - $W_{i,j}$ = "Wumpus at (i, j) (dead or alive)"
  - $GL_{i,j}$ = "Glitter at (i, j)"
  - $GO_{i,j}$ = "Gold at (i, j)"
- There are connections between some of them ...

Wumpus and propositional logic II

- Breezy and Pit:
  - A sq is breezy iff it is adj to a sq with a pit
  - A sq is smelly iff it is adj to a sq with a wumpus

  i.e.,
  - $B_{1,1} \leftrightarrow P_{1,2} \vee P_{2,1}$
  - $S_{1,1} \leftrightarrow W_{1,2} \vee W_{2,1}$
  - ...
- Note that these are true and therefore are in the KB K.
- Note that the breeze is what the agent can sense. According to the rules, the agent cannot sense a pit directly.

Wumpus and propositional logic III

- Glitter and gold:
  - A sq has a glitter iff it has a heap of gold

i.e.,

  - $GL_{1,1} \leftrightarrow GO_{1,1}$
  - ...

Wumpus and propositional logic IV

- Wumpus:
  - There is exactly one wumpus

  i.e.,
  - There is at least one wumpus
  - There is at most one wumpus

  i.e.,
  - $W_{1,1} \vee W_{1,2} \vee W_{1,3} \vee \cdots \vee W_{4,4}$
  - $\neg W_{1,1} \vee \neg W_{1,2}$
  - $\neg W_{1,1} \vee \neg W_{1,3}$
  - $\neg W_{1,1} \vee \neg W_{1,4}$
  - ...
  - $\neg W_{4,3} \vee \neg W_{4,4}$

Wumpus and propositional logic V

- Gold:
  - There is exactly one heap of gold

  Similar to above.

Wumpus and propositional logic VI

- "Location of agent" is time dependent
    - $L^t_{x,y}$ = "agent at location (x, y) at time t

  If the game runs from $t = 0$ up to $t = 99$, you'll need to create 16×100 location propositions.
- If time t runs from 0 to 99 (inclusive), you'll need 16×100 location propositions.
- Initially $L^0_{1,1}$ holds, i.e., $L^0_{1,1}$ is in the knowledge base K.
- **fluent** propositional variable = time dependent propositional variable with
- **atemporal** propositional variable = does not depend on time.

Wumpus and propositional logic VII

- Besides existence of exactly one wumpus, we also need to keep track of whether wumpus is alive or not which is time dependent:

$$\mathrm{WumpusAlive}^t$$

- At time 0, wumpus is alive

$$\mathrm{WumpusAlive}^0$$

holds, i.e., is in KB K.

Wumpus and propositional logic VIII

- Percept:
  - Percept is [stench sensor data, breeze sensor data, glitter sensor data, bump sensor data, scream sensor data]
  - Stench^t = "sense stench at time t"
  - Breeze^t = "sense breeze at time t"
  - Glitter^t = "sense glitter at time t"
  - Bump^t = "sense bump at time t"
  - Scream^t = "sense scream at time t"

Wumpus and propositional logic IX

- How to tie sensing an attribute of object at some time to the actual object at a location:

$$L^t_{x,y} \rightarrow (\text{Breeze}^{\underline{t}} \leftrightarrow B_{\underline{x,y}})$$

which represents "if agent is at location (x, y) at time t, it senses breeze at time t iff there's a breeze at (x, y)".

- Similarly

$$L^t_{x,y} \rightarrow (\text{Stench}^t \leftrightarrow S_{x,y})$$

Wumpus and propositional logic X

- Besides location, agent has direction and unit motion. And agent has arrow or does not have arrow.

  FacingEast^t, FacingWest^t, FacingNorth^t, FacingSouth^t
  HaveArrow^t

  At time 0,

  $$\text{FacingEast}^0, \text{HaveArrow}^0, \text{WumpusAlive}^0$$

  holds, i.e., they are in the knowledge base, K.

Wumpus and propositional logic XI

- Big picture: Recall that solving an AI problem is solving a search problem on a graph of states – transition diagram of states. Transition from one state to another is from action. Agent decides on action based on what's known:
  - from the knowledge base K at that time
  - from the percept at that time and at that location
- Viewing propositions as states, you can view implications are transitions of the state diagram. For instance

$$L_{1,1}^0 \wedge \text{FacingEast}^0 \wedge \text{Forward}^0 \rightarrow L_{2,1}^1 \wedge \neg L_{1,1}^1$$

which means "if agent is at location $(1,1)$ and time $t = 0$, and is facing east and executes move forward action, then at time $t = 1$, it is at location $(2,1)$ and not not at location $(1,1)$.

Wumpus and propositional logic XII

- Also need

$$\text{Forward}^0 \to (\text{HaveArrow}^0 \leftrightarrow \text{HaveArrow}^1)$$
$$\text{Forward}^0 \to (\text{WumpusAlive}^0 \leftrightarrow \text{WumpusAlive}^1)$$

- For the HaveArrow^t above, it can also be implemented as

$$\text{HaveArrow}^{t+1} \leftarrow \text{HaveArrow}^t \wedge \neg\,\text{Shoot}$$

- Exercise. Which method is better?

Wumpus and propositional logic XIII

- Exercise. Which method is better?
  Solution. The first method requires:

$$\begin{aligned}
\text{Forward}^0 &\rightarrow (\text{HaveArrow}^0 \leftrightarrow \text{HaveArrow}^1)\\
\text{TurnLeft}^0 &\rightarrow (\text{HaveArrow}^0 \leftrightarrow \text{HaveArrow}^1)\\
\text{TurnRight}^0 &\rightarrow (\text{HaveArrow}^0 \leftrightarrow \text{HaveArrow}^1)\\
\text{Climb}^0 &\rightarrow (\text{HaveArrow}^0 \leftrightarrow \text{HaveArrow}^1)\\
\text{Grab}^0 &\rightarrow (\text{HaveArrow}^0 \leftrightarrow \text{HaveArrow}^1)
\end{aligned}$$

Wumpus and propositional logic XIV

The second method requires

$$\text{HaveArrow}^{t+1} \leftarrow \text{HaveArrow}^{t} \wedge \neg \text{Shoot}$$

□

Wumpus and propositional logic XV

- Do the same for other action propositions:
- Action propositions:
  - Forward^t
  - TurnLeft^t
  - TurnRight^t
  - Grab^t
  - Shoot^t
  - Climb^t
- Example: $L^t_{x,y} \wedge \text{FacingEast}^t \wedge \text{Forward}^t \rightarrow L^{t+1}_{x+1,y}$

First order logic I

- Looking ahead ...
- First order logic (or predicate logic) is more expressive and allow us to model sentences of the following forms:
  - For all x, $\langle$proposition depending on $x\rangle$. Formally $\forall x(P(x))$.
  - There is some x such that $\langle$proposition depending on $x\rangle$. Formally $\exists x(P(x))$.
- Formally, "proposition depending on x" is a predicate.
- This then allows us to model propositions such as "every man has shot one monster" as

$$\forall x(\exists y(\mathrm{Man}(x) \wedge \mathrm{Monster}(y) \rightarrow \mathrm{Shot}(x, y)))$$