Artificial Intelligence

Propositional Logic

Playing with it in Python

Model

- Model?
  - Set of Boolean propositions
  - Set of assignments
- Common representations
  - { x, y }
    - Meaning a model where x is True and y is also True
    - Any other proposition not represented is assumed False
  - x/True y/True
    - Meaning a model where x is True, y is also True
    - No other propositions exist
- So:
  - { x, y } is set of assignments for the propositions { x, y, z }
    - Because, in this notation, omitted propositions are assumed False
  - X/True y/True is NOT a set of assignments for { x, y, z }
    - Because is makes not assignment to z

Is an expression/formula satisfiable?

- Yes, if there is a model that makes it True
- Finding if there is a model that makes an expression SATisfiable is the "SAT problem"

Model |= Satisfaction (SAT)

- Left side
  - Model / assignments
- Right side
  - Boolean expression
  - Logical formula
- Satisfaction?
  - If the assignments make the expression True
    - "the assignments satisfy the expression"
- Exercises with M = {x/True, y/False} - signal the satisfaction case(s)
  - M|=(x=>y)
  - M|=(x and y)
  - M|=y
  - M|=(x or y)

Model |= Satisfaction (SAT)

- (solution) Exercise with M = {x/True, y/False} - signal the satisfaction case(s)
  - M|=(x=>y)
  - M|=(x and y)
  - M|=y
  - M|=(x or y)

CNF = Conjunctive Normal Form

- Any propositional formula can be represented in CNF
- What is CNF?
  - ANDs of ORs, of literals
  - Literal? A variable or its negation
  - A formula in CNF is a conjunction of 1+ clause(s), each a disjunction of literals
  - Example:
    - (A OR B) AND (NOT A OR C) AND B
    - In John McCarthy's LISP notation:
      - (and (or A B) (or (not A) C) B)
- Why is CNF important?
  - Any propositional formula can be in CNF
  - The DPLL algorithm for the SAT problem operates on CNF

DPLL = Davis–Putnam–Logemann–Loveland

- What is the DPPL algorithm?
  - a complete, backtracking-based search algorithm for deciding the satisfiability of propositional logic formulas in CNF
- Recursive
  - Splits the problems into smaller sub-problems
  - Searches for the assignments that would make a formula satisfiable
- Some related concepts
  - "pure literal" - a Boolean var that appears with only one polarity (never negated or always negated)
  - "pure literal elimination" - pure literals can be assigned in a way that makes all clauses containing them true, so they do not constraint the search and can be eliminated
    - A form of simplification

Logical consequence (or "entailment")

- AKA Logical Implication
- Lexp 1, ... , Lexp n |= Rexp 1 , ... , Rexp n
  - L for "left"
  - R for "right"
  - Assume the , reads AND
- All the models that satisfy the left-side, must also satisfy the right-side
  - But the right-side might satisfy more models
- Exercise: which are logical consequences?
  - (p=>q),q |= p
  - (p and q) |= p
  - (p or q) |= p
  - (p or q), (not p) |= q
  - (p=>q), p |= q

Logical consequence

- (solution) Exercise: which are logical consequences?
  - (p=>q),q |= p
  - (p and q) |= p
  - (p or q) |= p
  - (p or q), (not p) |= q
  - (p=>q), p |= q

Logical equivalence

- L /// R
- L |= R
- R |= L
- Both must happen
- Exercise - signal the logical equivalence case(s)

$$((\neg u) \vee v) \equiv (\neg(u \wedge (\neg v)))$$

$$(a \wedge (b \vee c)) \equiv ((a \wedge b) \vee (a \wedge c))$$

$$((\neg x) \wedge (\neg y)) \equiv (\neg(x \wedge y))$$

$$((\neg x) \vee (\neg y)) \equiv (\neg(x \vee y))$$

$$((\neg u) \wedge v) \equiv (\neg(u \vee (\neg v)))$$

Logical equivalence

- (solution) Exercise - signal the logical equivalence case(s)
- @Left, equivalent
- @Right, not equivalent

$$((\neg u) \vee v) \equiv (\neg(u \wedge (\neg v)))$$

$$(a \wedge (b \vee c)) \equiv ((a \wedge b) \vee (a \wedge c))$$

$$((\neg x) \wedge (\neg y)) \equiv (\neg(x \wedge y))$$

$$((\neg x) \vee (\neg y)) \equiv (\neg(x \vee y))$$

$$((\neg u) \wedge v) \equiv (\neg(u \vee (\neg v)))$$

Python challenge

- Check companion file am_logical_helper.py
- Study companion class "LogicalHelper"
- Solve the exercises in these slides using an instance of the class