COMP219: Artificial Intelligence

Lecture 20: Propositional Reasoning

Overview

- Last time
  - Logic for KR in general; Propositional Logic; Natural Deduction
- Today
  - Entailment, satisfiability and validity
  - Normal forms
    - Negation normal form
    - Conjunctive normal form
  - Satisfiability as a search problem
    - DPLL algorithm
- Learning outcomes covered today:

  Distinguish the characteristics, and advantages and disadvantages, of the major knowledge representation paradigms that have been used in AI, such as production rules, semantic networks, propositional logic and first-order logic;

  Solve simple knowledge-based problems using the AI representations studied;

Propositional Logic for KR

- Given a knowledge base KB and a property α, check if $KB \models \alpha$
  - Model checking (e.g., use truth tables)
  - Theorem proving / inference: Prove α from KB
    - Natural deduction (last week)
    - Reduce to problems of *validity* or *unsatisfiability*
      - Davis-Putnam algorithm

Validity and Satisfiability

- A formula is said to be *valid*
  - *(or a **tautology**)*
  - iff it is true under ***every*** interpretation.
- A formula is said to be *satisfiable*
  - *(or **consistent**)*
  - iff it is true under ***at least one*** interpretation.
- A formula is said to be *unsatisfiable*
  - *(or **inconsistent** or **contradictory**)*
  - iff it is ***not*** made true under ***any*** interpretation.
- φ is a valid $\leftrightarrow$ $\neg\varphi$ is unsatisfiable.

Validity, Satisfiability and Entailment

Implications for Knowledge Representation…

- *Deduction Theorem:*
  KB $\models \alpha$ if and only if (KB $\Rightarrow \alpha$) is valid

Or. . .

- *Reductio ad absurdum:*
  KB $\models \alpha$ if and only if (KB $\wedge \neg\alpha$) is unsatisfiable

Satisfiability Checking

- Given a knowledge base KB and a property α, check if (KB $\wedge \neg\alpha$) is satisfiable
  - If not satisfiable, α is implied by KB
  - Otherwise, an interpretation of propositions would give us a *countermodel* – shows how (KB $\wedge \neg\alpha$) can be made true
- Compare Prolog… "?- goal(x)." ↔ "KB $\models$ goal(X)"
  - Prolog attempts to satisfy
    KB $\wedge$ goal(X) = KB $\wedge \neg(\neg$goal(X)$)$
    - not satisfiable: KB $\models \neg$goal(X)
    - satisfiable: countermodels to $\neg$goal(X), which are models of the goal.

Countermodel (I)

To check if

(hot ∧ smoky ⇒ fire)
∧(alarm_beeps ⇒ smoky)
∧(fire ⇒ switch_on_sprinklers)

?⊨

(alarm_beeps ⇒ switch_on_sprinklers)

we form

(hot ∧ smoky ⇒ fire)
∧(alarm_beeps ⇒ smoky)
∧(fire ⇒ switch_on_sprinklers)
∧¬(**alarm_beeps ⇒ switch_on_sprinklers**)

Countermodel (II)

But

(hot ∧ smoky ⇒ fire)
∧(alarm_beeps ⇒ smoky)
∧(fire ⇒ switch_on_sprinklers)
∧¬(**alarm_beeps ⇒ switch_on_sprinklers**)

is true under the interpretation *I*:

I(hot) = F
I(smoky) = T
I(fire) = F
I(alarm_beeps) = T
I(switch_on_sprinklers) = F

Example: Truth Tables and Satisfiability

Using a truth table show whether

$$(p \Rightarrow q) \vee (q \Rightarrow p)$$

is a tautology, consistent or inconsistent.

p	q	(p⇒q)	(q⇒p)	(p⇒q) ∨ (q⇒p)
T	T	T	T	T
T	F	F	T	T
F	T	T	F	T
F	F	T	T	T

Efficiency

- So we are back to model checking…
  - A truth table contains 2^n rows, construction requires 2^n steps. . .
- Can we do better than that?
  - Not in general: theory says that this is a *very* hard problem (Satisfiability checking is NP-complete)
  - In practice, not so bad, if we can use heuristics to identify lines we don't need to check
  - But we have to transform $(KB \wedge \neg\alpha)$ into a *normal form*

Equivalent Sentences

- Two sentences A and B are equivalent, written $A \equiv B$ iff A and B have the same truth values for every interpretation.
- Show

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

- Draw up a truth table for $(p \Rightarrow q)$ and $(\neg p \vee q)$ and check their truth values are the same.

p	q	(p⇒q)	(¬p)	(¬p∨q)
T	T	T	F	T
T	F	F	F	F
F	T	T	T	T
F	F	T	T	T

Equivalent Sentences

- Two sentences A and B are equivalent, written $A \equiv B$ iff A and B have the same truth values for every interpretation.
- Show

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

- Draw up a truth table for $(p \Rightarrow q)$ and $(\neg p \vee q)$ and check their truth values are the same.

≡ is not a 'logical operator' (as ⇔)

but similar to entailment…

In fact: two sentences are equivalent

$\alpha \equiv \beta$

if and only if

$\alpha \models \beta$ and $\beta \models \alpha$

(¬p∨q)
T
F
T
T

Equivalent Transformations (I)

- Where A, B and C are propositions or propositional formulae and T and F are true and false respectively
- Idempotent laws $A \wedge A \equiv A$
  $A \vee A \equiv A$
- Associative laws $(A \wedge B) \wedge C \equiv A \wedge (B \wedge C)$
  $(A \vee B) \vee C \equiv A \vee (B \vee C)$
- Commutative laws $A \wedge B \equiv B \wedge A$
  $A \vee B \equiv B \vee A$
- Distributive laws $A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$
  $A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$

Equivalent Transformations (II)

- Identity laws
  $A \wedge T \equiv A$
  $A \vee F \equiv A$
  $A \wedge F \equiv F$
  $A \vee T \equiv T$
- Complement laws
  $A \wedge \neg A \equiv F$
  $A \vee \neg A \equiv T$
  $\neg(\neg A) \equiv A$
  $\neg T \equiv F$
  $\neg F \equiv T$
- de Morgan's laws
  $\neg(A \wedge B) \equiv \neg A \vee \neg B$
  $\neg(A \vee B) \equiv \neg A \wedge \neg B$
- laws for $\Rightarrow$ and $\Leftrightarrow$
  $A \Rightarrow B \equiv \neg A \vee B$
  $A \Leftrightarrow B \equiv (A \Rightarrow B) \wedge (B \Rightarrow A)$
- We can use these laws to simplify expressions and to prove equivalences.

Examples

- Simplify the following expression: $\neg(\neg P \wedge \neg Q)$

$\neg(\neg P \wedge \neg Q)$	given
$\equiv (\neg\neg P \vee \neg\neg Q)$	de Morgan's laws
$\equiv (P \vee Q)$	Complement laws

- Prove the following equivalence: $\neg(\neg(P \wedge Q) \vee P) \equiv F$

$\neg(\neg(P \wedge Q) \vee P)$	given
$\equiv \neg((\neg P \vee \neg Q) \vee P)$	de Morgan's laws
$\equiv \neg((\neg Q \vee \neg P) \vee P)$	Commutative laws
$\equiv \neg(\neg Q \vee (\neg P \vee P))$	Associative laws
$\equiv \neg(\neg Q \vee T)$	Complement laws
$\equiv \neg T$	Complement laws
$\equiv F$	Complement laws

Negation Normal Form

- It is often useful to transform formulae into *normal forms*. These are logically equivalent formulae but have syntactically different forms that may be more suitable for reasoning with.
- There are several normal forms: Negation Normal Form, Clausal Form, Disjunctive Normal Form and Conjunctive Normal Form. We are most interested in **Conjunctive Normal Form (CNF).**
- but first: a formula is in *negation normal form* if negations appear only in front of propositions and the only operators are $\wedge$, $\vee$ and $\neg$.
  - First remove the $\Rightarrow$ and $\Leftrightarrow$ operators. Then apply de Morgan's laws and remove double negations (complement laws), until in the correct form.
  - $\wedge$, $\vee$ can still be nested

Conjunctive Normal Form

- A formula is in *Conjunctive Normal Form* if it is of the form
  $$A_1 \wedge A_2 \wedge \ldots A_k$$
  where each A_i is a **disjunction** of propositions or their negations.

- **Example**
  `(p ∨ q) ∧ r ∧ (¬p ∨ ¬r ∨ s)` is in CNF.
  `¬(p ∨ q) ∧ r ∧ (¬p ∨ ¬r ∨ s)` is not in CNF.
  `(p ∨ q) ∧ r ∧ (p ⇒ (¬r ∨ s))` is not in CNF.

Conjunctive Normal Form

- To translate into CNF, first translate into NNF. Then apply distribution laws or commutativity laws until in the correct form.

- **Any** well-formed formula of propositional logic can be rewritten, using the previous equivalences, as an **equivalent** formula of **CNF.**

Example

- Translate `(¬(p ∨ ¬q) ∨ r) ⇒ p` into CNF.
- We first translate into NNF and obtain

`¬(¬(p ∨ ¬q) ∨ r) ∨ p`
`((p ∨ ¬q) ∧ ¬r) ∨ p`

- Then transform into CNF

```
   ((p ∨ ¬q) ∧ ¬r) ∨ p
≡ ((p ∨ ¬q) ∨ p) ∧ (¬r ∨ p)
≡  (p ∨ ¬q ∨ p) ∧ (¬r ∨ p)
```


Exercise

- Convert the following into *Conjunctive Normal Form*, using the appropriate equivalence laws:

`p ⇔ (q ∨ r)`

Exercise

- Convert the following into *Conjunctive Normal Form*, using the appropriate equivalence laws:

$$p \Leftrightarrow (q \vee r)$$

$(p \Rightarrow (q \vee r)) \wedge ((q \vee r) \Rightarrow p)$	law for $\Leftrightarrow$
$(\neg p \vee q \vee r) \wedge (\neg (q \vee r) \vee p)$	law for $\Rightarrow$
$(\neg p \vee q \vee r) \wedge ((\neg q \wedge \neg r) \vee p)$	de Morgan's law
$(\neg p \vee q \vee r) \wedge (\neg q \vee p) \wedge (\neg r \vee p)$	distributive law

Satisfiability as a Search Problem

- Given a formula in CNF, can we find an assignment of truth values to propositions that satisfies it?

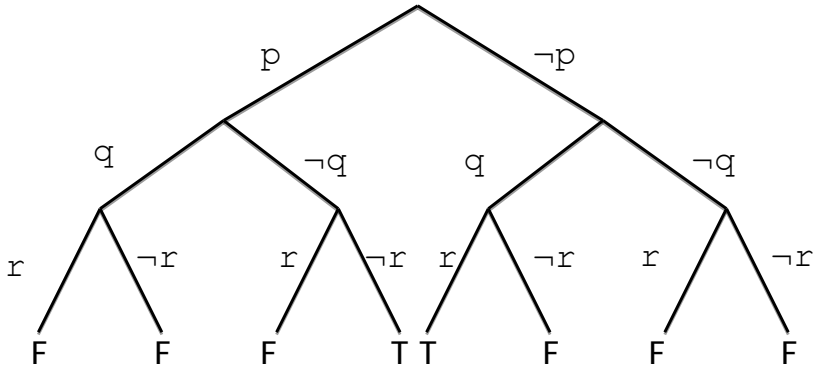
- States are *partial assignments* - some propositions get values, some are possibly unassigned.
- Actions are deciding whether a (yet unassigned) proposition is true or false.
- Initial state: empty partial assignment.
- Goal state: an assignment making the formula true.

Algorithm as Satisfiability for CNF

- A complete backtracking algorithm – Davis-Putnam paper (1960)
- The version presented (**DPLL**) is described in a paper by Davis, Logemann, and Loveland (1962).
- Naive Approach: every proposition is either true or false in any interpretation.

Search Space

- Consider $(\neg p \vee \neg r) \wedge (p \vee q) \wedge (r \vee \neg q)$



Every path in the tree represents a (partial) assignment

Partial Assignment

- Another idea: Simplify formula with a (previously guessed) partial assignment

  `(¬p ∨ ¬r) ∧ (p ∨ q) ∧ (r ∨ ¬q)`

{ if we tried `p = True` we can now simplify: }

  `(¬True ∨ ¬r) ∧ (True ∨ q) ∧ (r ∨ ¬q)`

  `(False ∨ ¬r) ∧ True ∧ (r ∨ ¬q)`

  `¬r ∧ (r ∨ ¬q)`

{ can only be true if `r = False` }

  `¬False ∧ (False ∨ ¬q)`

  `¬q`

{ can only be true if `¬q = True` (so `q = False`) }

  `True`

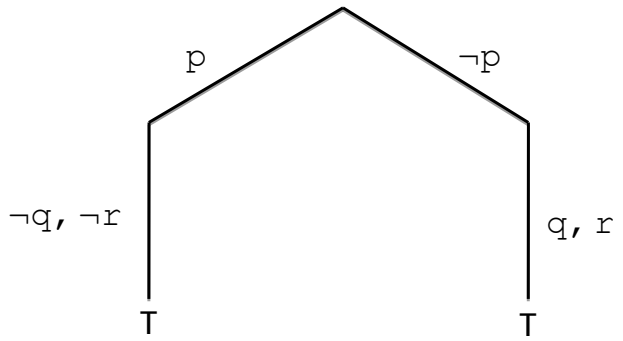
- Given a good order for partial assignments this can be very helpful. DPLL provides heuristics for choosing partial assignments. Only in the worst case we will need to try everything.

Likewise

- `(¬p ∨ ¬r) ∧ (p ∨ q) ∧ (r ∨ ¬q)`
  { let us now consider `p = False` }
- `(¬False ∨ ¬r) ∧ (False ∨ q) ∧ (r ∨ ¬q)`
- `(True ∨ ¬r) ∧ q ∧ (r ∨ ¬q)`
- `True ∧ q ∧ (r ∨ ¬q)`
- `q ∧ (r ∨ ¬q)`
  { can only be true if `q = True` }
- `True ∧ (r ∨ ¬True)`
- `(r ∨ False)`
  { `r` can only be true if `r = True` }
- `True`

Reduced Search Space

- Consider `(¬p ∨ ¬r) ∧ (p ∨ q) ∧ (r ∨ ¬q)`



Only 5 nodes!

Algorithm Structure

- Search through possible assignments of propositions
- Simplify formulae with partial assignments
  - Unit clause propagation
  - Pure literal elimination

Unit Clause

- A clause with just one literal is called *a unit clause*
  - e.g. $\boxed{q} \wedge (\neg r \vee \neg q) \wedge (r \vee s)$
- Literal's value can be *uniquely* assigned
  - `q` **must** be set to `True`

- Unit clause propagation:
  Check if a formula in CNF has a unit clause, `C`.
  Set the value of the literal in `C` such that `C` is `True`
  - Notice that some other clauses may become unit
  - e.g. $\boxed{q} \wedge (\neg r \vee \neg q) \wedge (r \vee s)$ reduces to $\neg r \wedge (r \vee s)$
  - r **must** be set to `False`
  - $\boxed{\neg r} \wedge (r \vee s)$ reduces to `s`

Pure Literal

- A *pure literal* is a literal that always appears with the same "sign" in all clauses.
  - e.g. $(\boxed{p} \vee q) \wedge (\neg q \vee \neg r) \wedge (\neg q \vee r)$
- Making a pure literal `True` makes *some* clauses `True`, but *no* clause `False`
  - e.g. $(p \vee q) \wedge (\neg q \vee \neg r) \wedge (\neg q \vee r)$ reduces to $(\boxed{\neg q} \vee \neg r) \wedge (\boxed{\neg q} \vee r)$
  - $(\neg q \vee \neg r) \wedge (\neg q \vee r)$ reduces to `True`
- Pure literal elimination:
  Check if a formula in CNF has a pure literal, l
  Set the value of l to `True`

DPPL(φ)

- Given a propositional formula φ, DPLL(φ) does the following:
- If φ is `True` then return True;
- If φ is `False` then return False;
- Pick a symbol `p` in φ
  - If DPLL(Simplify(φ, `p`)) is `True` then return True;
  - If DPLL(Simplify(φ, $\neg$`p`)) is `True` then return True;
- Return False;

Simplify(φ, l)

- Given a propositional formula φ and a literal l, $simplify(\varphi, l)$ does the following:
  - Delete every clause that contains l from φ;
  - Delete $\neg l$ from all clauses in φ
  - Apply exhaustively unit clause propagation and pure literal elimination;
  - Fail (return False) if positive **and** negative unit clauses for the **same** literal.

Applications

- There are now several efficient SAT solvers based on DPLL available on the web. They are used in a number of applications.
  - Hardware and software verification
    - IBM, Intel, . . .
  - Planning
  - Scheduling
  - . . .

Summary

- We have considered issues concerning efficiency in propositional reasoning
- This has covered equivalent transformations and normal forms
  - Negation normal form
  - Conjunctive normal form
- We have considered how satisfiability can be viewed as a search problem
  - Looked at an algorithm for satisfiability in CNF

- Next time
  - Proof method *resolution* for propositional logic