

COMP3411: Artificial Intelligence

Automated Reasoning

C. Sammut & W. Wobcke

This Lecture

- Proof systems
 - ▶ Soundness, completeness, decidability
- Resolution and Refutation
- Horn clauses and SLD resolution
- Prolog

Summary So Far

- Propositional Logic
 - ▶ Syntax: Formal language built from $\wedge, \vee, \neg, \rightarrow$
 - ▶ Semantics: Definition of truth table for every formula
 - ▶ $S \models P$ if whenever all formulae in S are True, P is True
- Proof System
 - ▶ System of axioms and rules for **deduction**
 - ▶ Enables computation of proofs of P from S
- Basic Questions
 - ▶ Are the proofs that are computed always correct? (soundness)
 - ▶ If $S \models P$, is there always a proof of P from S (completeness)

Mechanising Proof

- A **proof** of a formula P from a set of **premises** S is a sequence of lines in which any line in the proof is
 1. An axiom of logic or premise from S , or
 2. A formula deduced from previous lines of the proof using a **rule of inference** and the last line of the proof is the formula P
- Formally captures the notion of mathematical proof
- S **proves** P ($S \vdash P$) if there is a proof of P from S ; alternatively, P **follows** from S
- Example: Resolution proof

Soundness and Completeness

- A proof system is **sound** if (intuitively) it preserves truth
 - ▶ Whenever $S \vdash P$, if every formula in S is True, P is also True
 - ▶ Whenever $S \vdash P$, $S \models P$
 - ▶ If you start with true assumptions, any conclusions **must** be true
- A proof system is **complete** if it is capable of proving all consequences of any set of premises (including infinite sets)
 - ▶ Whenever P is entailed by S , there is a proof of P from S
 - ▶ Whenever $S \models P$, $S \vdash P$
- A proof system is **decidable** if there is a mechanical procedure (computer program) which when asked whether $S \vdash P$, can **always** answer ‘true’ – or ‘false’ – correctly

Resolution

- A common type of proof system based on **refutation**
- Better suited to computer implementation than systems of axioms and rules (**can** give correct 'false' answers)
- Decidable in the case of Propositional Logic
- Generalises to First-Order Logic (see next set of lectures)
- Needs all formulae to be converted to **clausal form**

Normal Forms

- A **literal** ℓ is a propositional variable or the negation of a propositional variable (P or $\neg P$)
- A **clause** is a disjunction of literals $\ell_1 \vee \ell_2 \vee \cdots \vee \ell_n$
- Conjunctive Normal Form (CNF) — a conjunction of clauses, e.g.

$(P \vee Q \vee \neg R) \wedge (\neg S \vee \neg R)$ – or just one clause, e.g. $P \vee Q$

- Disjunctive Normal Form (DNF) — a disjunction of conjunctions of literals, e.g.

$(P \wedge Q \wedge \neg R) \vee (\neg S \wedge \neg R)$ – or just one conjunction, e.g. $P \wedge Q$

- Every Propositional Logic formula can be converted to CNF and DNF
- Every Propositional Logic formula is equivalent to its CNF and DNF

Conversion to Conjunctive Normal Form

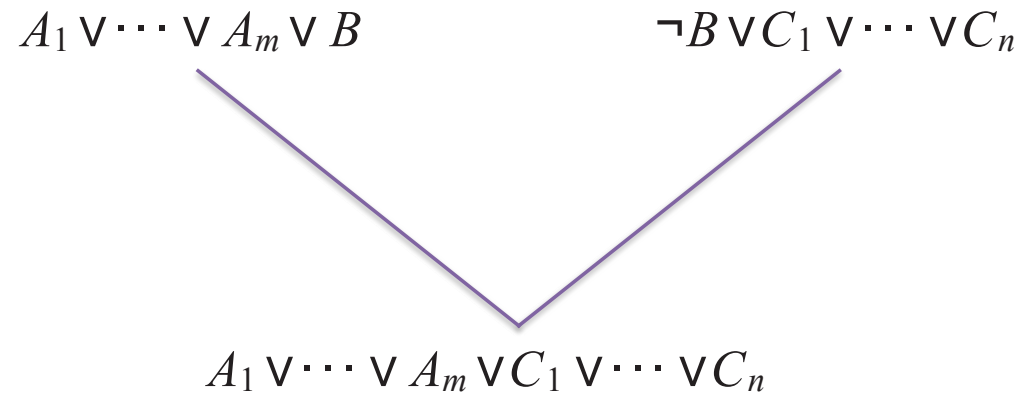
- Eliminate \leftrightarrow rewriting $P \leftrightarrow Q$ as $(P \rightarrow Q) \wedge (Q \rightarrow P)$
- Eliminate \rightarrow rewriting $P \rightarrow Q$ as $\neg P \vee Q$
- Use De Morgan's laws to push \neg inwards (repeatedly)
 - ▶ Rewrite $\neg(P \wedge Q)$ as $\neg P \vee \neg Q$
 - ▶ Rewrite $\neg(P \vee Q)$ as $\neg P \wedge \neg Q$
- Eliminate double negations: rewrite $\neg \neg P$ as P
- Use the distributive laws to get CNF [or DNF] – if necessary
 - ▶ Rewrite $(P \wedge Q) \vee R$ as $(P \vee R) \wedge (Q \vee R)$ [for CNF]
 - ▶ Rewrite $(P \vee Q) \wedge R$ as $(P \wedge R) \vee (Q \wedge R)$ [for DNF]

Example Clausal Form

Clausal Form = set of clauses in the CNF

- $\neg(P \rightarrow (Q \wedge R))$
- $\neg(\neg P \vee (Q \wedge R))$
- $\neg\neg P \wedge \neg(Q \wedge R)$
- $\neg\neg P \wedge (\neg Q \vee \neg R)$
- $P \wedge (\neg Q \vee \neg R)$
- Clausal Form: $\{P, \neg Q \vee \neg R\}$

Resolution Rule of Inference



where B is a propositional variable and A_i and C_j are literals

- B and $\neg B$ are **complementary literals**
- $A_1 \vee \cdots \vee A_m \vee C_1 \vee \cdots \vee C_n$ is the **resolvent** of the two clauses
- Special case: If no A_i and C_j , resolvent is empty clause, denoted \square or \perp

Resolution Rule

- Consider $A_1 \vee \cdots \vee A_m \vee B$ and $\neg B \vee C_1 \vee \cdots \vee C_n$
 - ▶ Suppose both are True
 - ▶ If B is True, $\neg B$ is False so $C_1 \vee \cdots \vee C_n$ must be True
 - ▶ If B is False, $A_1 \vee \cdots \vee A_m$ must be True
 - ▶ Hence $A_1 \vee \cdots \vee A_m \vee C_1 \vee \cdots \vee C_n$ is True

Hence the resolution rule is **sound**

- Starting with true premises, any conclusion made using resolution **must** be true

Applying Resolution: Naive Method

- Convert knowledge base into clausal form
- Repeatedly apply resolution rule to the resulting clauses
- P follows from the knowledge base if and only if each clause in the CNF of P can be derived using resolution from the clauses of the knowledge base (or subsumption)
- Example
 - ▶ $\{P \rightarrow Q, Q \rightarrow R\} \vdash P \rightarrow R$
 - ▶ Clauses $\neg P \vee Q, \neg Q \vee R$, show $\neg P \vee R$
 - ▶ Follows from one resolution step (Q and $\neg Q$ cancel, leaving $\neg P \vee R$)

Refutation Systems

- To show that P follows from S (i.e. $S \vdash P$) using [refutation](#), start with S and $\neg P$ in clausal form and derive a contradiction using resolution
- A contradiction is the “empty clause” (a clause with no literals)
- The empty clause \square is unsatisfiable (always False)
- So if the empty clause \square is derived using resolution, the original set of clauses is unsatisfiable (never all True together)
- That is, if we can derive \square from the clausal forms of S and $\neg P$, these clauses can never be all True together
- Hence whenever the clauses of S are all True, at least one clause from $\neg P$ must be False, i.e. $\neg P$ must be False and P must be True
- By definition, $S \models P$ (so P can correctly be concluded from S)

Applying Resolution Refutation

- Negate query to be proven (resolution is a refutation system)
- Convert knowledge base and negated query into CNF
- Repeatedly apply resolution until either the empty clause (contradiction) is derived or no more clauses can be derived
- If the empty clause is derived, answer 'true' (query follows from knowledge base), otherwise answer 'false' (query does not follow from knowledge base)

Resolution: Example 1

$(G \vee H) \rightarrow (\neg J \wedge \neg K), G \vdash \neg J$

Clausal form of is $\{\neg G \vee \neg J, \neg H \vee \neg J, \neg G \vee \neg K, \neg H \vee \neg K\}$

1. $\neg G \vee \neg J$ [Premise]

2. $\neg H \vee \neg J$ [Premise]

3. $\neg G \vee \neg K$ [Premise]

4. $\neg H \vee \neg K$ [Premise]

5. G [Premise]

6. J [\neg Query]

7. $\neg G$ [1, 6 Resolution]

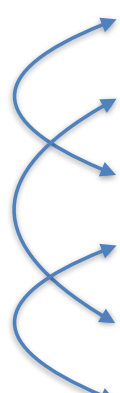
8. \square [5, 7 Resolution]

Resolution: Example 2

$$P \rightarrow \neg Q, \neg Q \rightarrow R \vdash P \rightarrow R$$

Recall $P \rightarrow R \Leftrightarrow \neg P \vee R$





Clausal form of $\neg(\neg P \vee R)$ is $\{P, \neg R\}$

- 
- 1. $\neg P \vee \neg Q$ [Premise]
 - 2. $Q \vee R$ [Premise]
 - 3. P [\neg Query]
 - 4. $\neg R$ [\neg Query]
 - 5. $\neg Q$ [1, 3 Resolution]
 - 6. R [2, 5 Resolution]
 - 7. \square [4, 6 Resolution]

Resolution: Example 3

$$\vdash ((P \vee Q) \wedge \neg P) \rightarrow Q$$

Clausal form of $\vdash ((P \vee Q) \wedge \neg P) \rightarrow Q$ is $\{P \vee Q, \neg P, \neg Q\}$

- | | | |
|--|---------------|-----------------------------|
|  | 1. $P \vee Q$ | $[\neg \text{Query}]$ |
|  | 2. $\neg P$ | $[\neg \text{Query}]$ |
|  | 3. $\neg Q$ | $[\neg \text{Query}]$ |
|  | 4. Q | $[1, 2 \text{ Resolution}]$ |
| | 5. \square | $[3, 4 \text{ Resolution}]$ |

Rewriting negated query in CNF:

$$\neg [((P \vee Q) \wedge \neg P) \rightarrow Q]$$

$$\neg [\neg ((P \vee Q) \wedge \neg P) \vee Q]$$

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

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

Now write in clausal form:

$$\{P \vee Q, \neg P, \neg Q\}$$

Soundness and Completeness Again

For Propositional Logic

- Resolution refutation is **sound**, i.e. it preserves truth (if a set of premises are all true, any conclusion drawn from those premises **must** also be true)
- Resolution refutation is **complete**, i.e. it is capable of proving all consequences of any knowledge base (not shown here!)
- Resolution refutation is **decidable**, i.e. there is an algorithm implementing resolution which when asked whether $S \vdash P$, can always answer ‘true’ or ‘false’ (correctly)

Heuristics in Applying Resolution

- Clause elimination — can disregard certain types of clauses
 - ▶ Pure clauses: contain literal L where $\neg L$ doesn't appear elsewhere
 - ▶ Tautologies: clauses containing both L and $\neg L$
 - ▶ Subsumed clauses: another clause is a subset of the literals
- Ordering strategies
 - ▶ Resolve unit clauses (only one literal) first
 - ▶ Start with query clauses
 - ▶ Aim to shorten clauses

Horn Clauses

Using a less expressive language makes proof procedure easier.

- Review
 - ▶ **literal** – proposition variable or negation of proposition variable
 - ▶ **clause** – disjunction of literals
- **Definite Clause** – exactly one positive literal
 - ▶ e.g. $B \vee \neg A_1 \vee \dots \vee \neg A_n$, i.e. $B \leftarrow A_1 \wedge \dots \wedge A_n$
- **Negative Clause** – no positive literals
 - ▶ e.g. $\neg Q_1 \vee \neg Q_2$ (negation of a query)
- **Horn Clause** – clause with at most one positive literal

Prolog

- Horn clauses in First-Order Logic
- SLD resolution
- Depth-first search strategy with backtracking
- User control
 - ▶ Ordering of clauses in Prolog database (facts and rules)
 - ▶ Ordering of subgoals in body of a rule
- Prolog is a programming language based on resolution refutation relying on the programmer to exploit search control rules

Prolog Clauses

$P :- Q, R, S.$

$P \leftarrow Q \wedge R \wedge S.$

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

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

Prolog DB = set of clauses

Queries:

?- Q, R, S

$\perp \leftarrow Q \wedge R \wedge S$

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

$\neg Q \vee \neg R \vee \neg S$

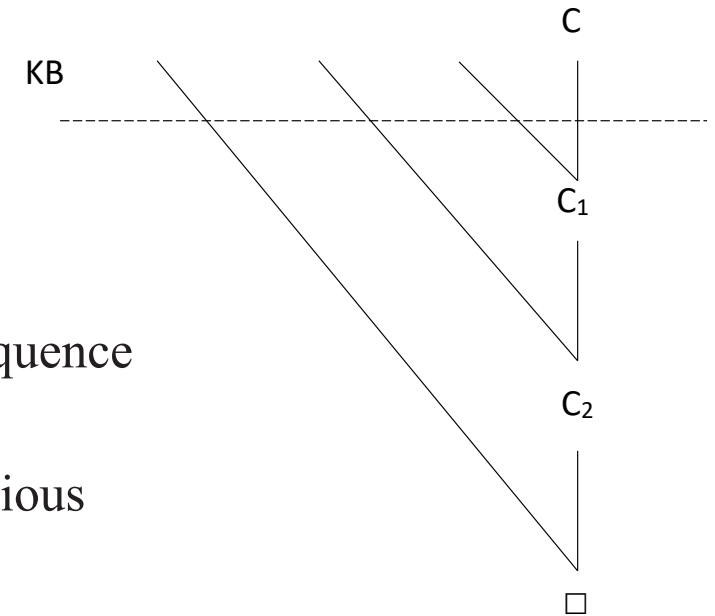
$$P \rightarrow Q \equiv \neg P \vee Q$$

$$P \leftarrow Q \equiv P \vee \neg Q$$

$$\perp \equiv \text{false (i.e. a contradiction)}$$

SLD Resolution – \vdash_{SLD}

- Selected literals **L**inear form **D**efinite clauses resolution
- SLD refutation of a clause C from a set of clauses KB is a sequence
 1. First clause of sequence is C
 2. Each intermediate clause C_i is derived by resolving the previous clause C_{i-1} and a clause from KB
 3. The last clause in the sequence is \square
- For a definite KB and negative clause query Q : $KB \cup Q \vdash \square$
if and only if $KB \cup Q \vdash_{SLD} \square$



Prolog Example

```
r.                % facts
u.
v.

q :- r, u.        % rules
s :- v.
p :- q, r, s.

?- p.             % query
true
```


Example Execution of Prolog interpreter

`r.`

`u.`

`v.`

`q :- r, u.`

`s :- v.`

`p :- q, r, s.`

`?- p.`

Initial goal set = {p}

1. {q, r, s}

because p :- q, r, s.

2. {r, u, r, s}

because q :- r, u.

3. {u, r, s}

because r.

4. {r, s}

because u.

5. {s}

because r.

6. {v}

because s :- v

7. {}

because v.

8. => true

because empty clause

- In each step, we remove the first element in the goal set and replace it with the body of the clause whose head matches that element. E.g. remove *p* and replace by *q, r, s*.
- **Note:** The simple Prolog interpreter isn't smart enough to remove the duplication of *r* in step 2.

Prolog Interpreter

Input: A query Q and a logic program KB

Output: 'true' if Q follows from KB , 'false' otherwise

Initialise current goal set to $\{Q\}$

while the current goal set is not empty do

Choose G from the current goal set; (first in goal set)

Make a copy $G' :- B_1, \dots, B_n$ of a clause from KB

(try all in KB) (if no such rule, try alternative rules)

Replace G by B_1, \dots, B_n in current goal set

if current goal set is empty:

output 'true'

else output 'false'

Inefficient and not how a
real Prolog interpreter works

- Depth-first, left-right with backtracking

Conclusion: Propositional Logic

- Propositions built from \wedge , \vee , \neg , \rightarrow
- Sound, complete and decidable proof systems (inference procedures)
 - ▶ Natural deduction
 - ▶ Resolution refutation
 - ▶ Prolog for special case of definite clauses
 - ▶ Tableau method
- Limited expressive power
 - ▶ Cannot express ontologies (no relations)
- **First-Order Logic** can express knowledge about objects, properties and relationships between objects