Knowledge Representation and Reasoning

Propositional Calculus
and Resolution

Propositional Calculus

- What is Propositional Calculus (P.C.)?
- Elements of P.C. – Formal Language
- Reasoning: Proofs and Entailments
- Clauses and Resolution
- Resolution as Inference Rule
- Algorithms and Complexity

Formal P.C. Language

Elements of P.C. language:

- Atoms: P, R, Q, A2, ...
- Connectives: $\wedge$, $\vee$, $\supset$, $\neg$ (AND, OR, IMPLIES, NOT)
- Well-formed formulas (**wff**)

  P , $R \wedge A1$, $P1 \supset (\neg B)$, ...

Formal P.C. Language

The Propositional Truth Table:

P1	P2	$P1 \wedge P2$	$P1 \vee P2$	$\neg P1$	$P1 \supset P2$
True	True	True	True	False	True
True	False	False	True	False	False
False	True	False	True	True	True
False	False	False	False	True	True

Note: Semantics of "implies" ($P1 \supset P2$) is equivalent to ($\neg P1 \vee P2$)

Formal P.C. Language

Rules of Inference:

- $\{P1 , P2\} \rightarrow P1 \wedge P2$
- $\{P1\} \rightarrow P1 \vee (any)$
- $\{P1 , (P1 \supset P2)\} \rightarrow P2$ *“modus ponens”*
- $\{\neg \neg P1\} \rightarrow P1$
- ...

Formal P.C. Language

Proofs:

D = sequence of wff (prior knowledge)

- E = "theorem", a wff to be proven
- Proof: D $\rightarrow$ E (**deduction** rule)
- Example: D={P,R,P⊃Q} , E={Q∧R}

  D={P,R,P⊃Q}→{P,P⊃Q,Q,R,Q ∧R}→{Q ∧R}=E

Entailment:

- "Discover" wff that hold true given D

Formal P.C. Language

Equivalence of wff:

- Produce the same outcome in all cases
- Formal definition:

  $P1 \equiv P2 \ : \ (P1 \supset P2) \wedge (P2 \supset P1)$

Formal P.C. Language

<u>Soundness</u>:

- If any P1 that is implied by D and rules R can be "discovered" by entailement

<u>Completeness</u>:

- If any P1 that is implied by D and rules R can be "proven" by a proof

- The PSAT (Propositional Satisfiability) Problem: find a model D that implies a given formula P1
- Common situation in circuit design, path planning, etc.
- Usual form: **CNF – Conjunctive Normal Form**

Resolution in P.C.

- Resolution: use deduction rules to assert or discard the validity of a Clause.
- Clause: any formatted wff that is used in a Resolution scheme.
- Resolution on Clauses:
  - Follow **3 simple rules** for converting any wff into a clause in CNF that can be resolved

Resolution in P.C.

- Converting a wff to a clause (CNF):

  $\neg(P \supset Q) \vee (R \supset P)$

1. Eliminate implication signs ($\supset$) by using the equivalent form using ($\neg\vee$):

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

2. Reduce the ($\neg$) signs using De Morgan's laws:

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

3. Convert to CNF, i.e. place ($\wedge$) outside parentheses:

   $(P \vee \neg R \vee P) \wedge (\neg Q \vee \neg R \vee P) \equiv \{(P \vee \neg R) , (\neg Q \vee \neg R \vee P)\}$

Resolution in P.C.

Resolution as Inference Rule:

- Resolution is “sound” BUT not “complete”, i.e. not all logical expressions can be entailed.
- Instead, formulate the negation of the clause to be entailed and then investigate if this new clause can be proven within the current “state of world”.
- Resolution Refutation: proof of the negation (non-empty result) invalidates the original clause, otherwise the original clause is asserted as true.
- Note: Resolution with CNF is faster, since it can be finalized when one term gets invalidated (false).

P.C. Resolution Strategies

Problem: In what **order** should the resolutions be performed for optimal results?

- **Breadth-first**: expand all nodes in same level
- **Depth-first**: expand each node to the end
- **Unit-preference**: expand "small" nodes first

- Horn clauses: contain at most one positive literal
  $\Rightarrow$ limits the complexity of deduction search to linear times.

Exercices – Learning Objectives

- Learn to describe a real-world problem with a set of Rules and desired Functionality
- Translate specifications into Predicates
- Translate Predicates into Truth Tables
- Bottom-Up approach: implement Truth Tables using Boolean expressions
- Top-Down approach: implement Predicates (CNF) as Boolean expressions

Exercise #1: 3x8 multiplexer

- Design a Boolean implementation of a 3x8 multiplexer chip with functionality as described in the following Truth Table:

A0	A1	A2	B0	B1	B2	B3	B4	B5	B6	B7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Exercise #2: Simple ABS controller

- Design a Boolean implementation of a simplified anti-blocking system (ABS) for car brakes, according to the following logical functionality:

- Car brakes work in two modes: ABS "ON" and ABS "OFF"
- ABS is activated when brake pedal is pressed and wheel(s) is sliding on the road, instead of rolling.
- ABS controller should be activated for opposing pair(s) of wheels, i.e. for both wheels on the same axis (front pair, rear pair or all four wheels)

- Hint:
  - Begin by first formulating ABS functionality for each wheel using a predicate expression like:

ABS(Wheel_FR , "ON") = BRAKE(Wheel_FR , "ON") $\wedge \neg$ SLIDING(Wheel_FR)

P.C. – Readings

- Nils J. Nilsson, “Artificial Intelligence – A New Synthesis”, Morgan Kaufmann Publishers (1998). [see: ch.13 & ch.14]

- S. J. Russell, P. Norvig, “Artificial Intelligence: A Modern Approach”, 2nd/Ed, Prentice Hall, 2002.