P. Blackburn and J. Bos: “Representation and Inference for Natural Language“ – Ch. 4

# Propositional Resolution

Andreas Rudin

Frank Richter, Fritz Hamm:
“Algorithmic Semantics”

May 6, 2010

# Outline

- Conjunctive Normal Form
  - Terminology
  - Set CNF
  - Transformation
- Resolution Rule
  - Terminology
  - Usage in Theorem Proving
- Prolog Implementation
  - Structure
  - Converting formulas into set CNF
  - Performing Resolution
- Conclusion

# Introduction

- Informativity check:
  Is a formula "capable" of delivering any information?
- Refutation method (reductio ad impossibile):
  If the negation has a contradiction (i.e. is always false) – the original formula is always true = valid.
- Another example: Tableau-Method

# Conjunctive Normal Form Terminology

- Literals
  - Positive Literals: p, q, r
  - Negative Literals: ¬p, ¬q, ¬r
- Clauses
  - Disjunction of literals: p∨¬q∨r
  - Can be written as: [p, ¬q, r]
  - If at least one of the literals is true, the clause is true.
  - Empty clauses are always false: [] = ⊥

# Conjunctive Normal Form Terminology

- Conjunctive Normal Form (CNF)
  - A formula is in CNF iff it is a conjunction of clauses:

    $(p \lor \neg q) \land (q \lor r) \land (r \lor \neg s \lor \neg q)$

  - Can also be written as:

    [[p,¬q],[q,r],[r,¬s,¬q]]

  - If a formula in CNF contains an empty clause, it is unsatisfiable (i.e. can never be true)

# Conjunctive Normal Form
## Set CNF

- A Formula is in set CNF if no clause:
  - occurs more than once:

    [[¬s,p],[r,¬q,t],[r,¬q,s],[p,¬s]]

  - contains a repeated literal:

    [[¬r],[p,¬t,q,p,s],[¬s,¬t,¬r]]

# Conjunctive Normal Form Transformation

- First: Transformation to NNF (Negation Normal Form)

| Transformation to NNF Rules | |
|---|---|
| 1) $\neg(\varphi \wedge \psi)$ ▶ $\neg\varphi \vee \neg\psi$ | 3) $\neg(\varphi \rightarrow \psi)$ ▶ $\varphi \wedge \neg\psi$ |
| 2) $\neg(\varphi \vee \psi)$ ▶ $\neg\varphi \wedge \neg\psi$ | 4) $\varphi \rightarrow \psi$ ▶ $\neg\varphi \vee \psi$ |
| 5) $\neg\neg\varphi$ ▶ $\varphi$ | |

- Second: Transformation to CNF

  By repetitive applications of the distributive and associative rules.

| Distributive Rules | Associative Rules |
|---|---|
| $\theta \vee (\varphi \wedge \psi)$ ▶ $(\theta \vee \varphi) \wedge (\theta \vee \psi)$ | $(\varphi \wedge \psi) \wedge \theta$ ▶ $\theta \wedge (\varphi \wedge \psi)$ |
| $(\varphi \wedge \psi) \vee \theta$ ▶ $(\varphi \vee \theta) \wedge (\psi \vee \theta)$ | $(\varphi \vee \psi) \vee \theta$ ▶ $\theta \vee (\varphi \vee \psi)$ |

# Resolution Rule
## Terminology

- Complementary structures:
  - Complementary pairs:
    A positive literal in one clause and its negation in another clause are a *complementary pair*, also called *resolvents*
  - Complementary clauses:
    Those clauses that contain c.-pairs are *complementary clauses*.

[[p,¬t],[q,p],[t,¬s]]

# Resolution Rule
## Terminology

- Binary Resolution Rule:

$$[p_1,\ldots,p_n,{\color{red}r},p_{n+1},p_m] \text{ and } [q_1,\ldots,q_n,{\color{red}\neg r},q_{n+1},q_m]$$



$$[p_1,\ldots,p_n,p_{n+1},p_m,q_1,\ldots,q_n,q_{n+1},q_m]$$

  – Explanation:
    Since we know that clauses are disjunctions and r and ¬r can‘t be true at the same time, we can be certain that at least one of the other literals must be true.


Propositional Resolution

Conjunctive Normal Form

Resolution Rule

Implementation

Conclusion

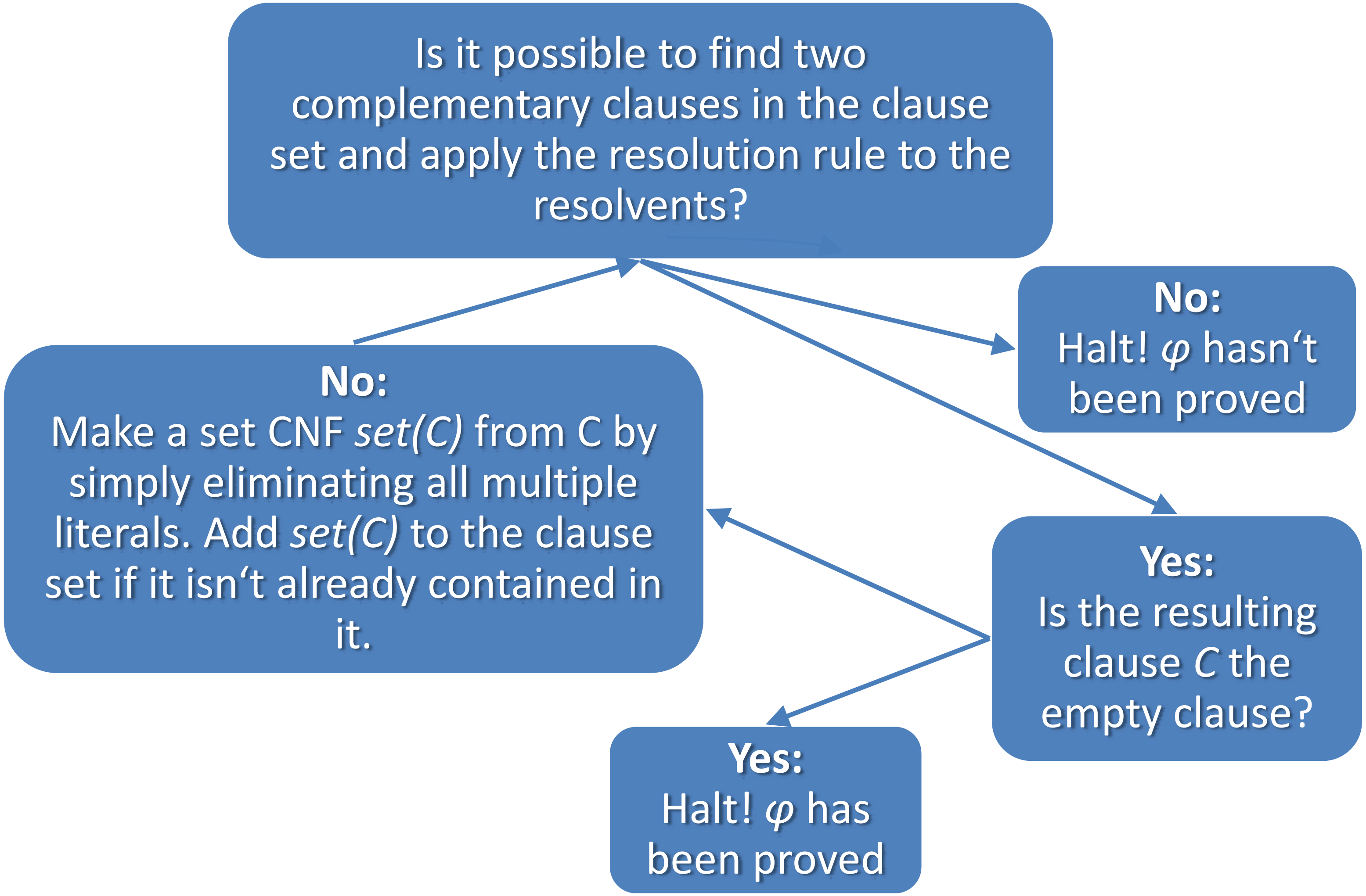
# Resolution Rule

## Usage in Theorem Proving

- Refutation:
  In order to show the validity of $\varphi$, we try to show a contradiction in $\neg\varphi$
- The contradiction becomes apparent when we can show that $\neg\varphi$ in CNF contains an empty clause ($\perp$)
- Two-step process:

  $\neg\varphi$ → CNF → Repeatedly use the Resolution Rule to generate an empty clause.

# Resolution Rule
## Usage in Theorem Proving



# Prolog Implementation

## Structure

1. Converting formulas into set CNF

```
cnf(Formula, SetCNF):-
   nnf(Formula,NNF),
   nnf2cnf([[NNF]],[],CNF),
   setCnf(CNF,SetCNF).
```

2. Performing Resolution

```
rprove(Formula):-
   cnf(not(Formula),CNF),
   Refute(CNF).
```

# Prolog Implementation
## Converting formulas into set CNF

| Transformation to NNF Rules | |
|---|---|
| 1) ¬(φ∧ψ) ▶ ¬φ∨¬ψ | 3) ¬(φ→ψ) ▶ φ∧¬ψ |
| 2) ¬(φ∨ψ) ▶ ¬φ∧¬ψ | *4)* φ→ψ ▶ ¬φ∨ψ |
| 5) ¬¬φ ▶ φ | |

Rule 1):

```
nnf(not(and(F1,F2)),or(N1,N2)):-
    nnf(not(F1),N1),
    nnf(not(F2),N2).
```

Rule 3):

```
nnf(not(imp(F1,F2)),and(N1,N2)):-
    nnf(F1,N1),
    nnf(not(F2),N2).
```

# Prolog Implementation
## Performing Resolution

Input C is the negation of a formula in set CNF.

```
refute(C):-
   \+ memberList([],C)
   resolveList(C,[],Output),
   unionSets(Output,C,NewC),
   \+ NewC = C,
   refute(NewC).
```

- `\+ memberList([],C)` — Is there a [] in C? Continue if not
- `resolveList(C,[],Output)` — Resolve and add all possible solutions to Output
- `unionSets(Output,C,NewC)` — Unite Output and C to NewC (NewC must be made a set if needed!)
- `\+ NewC = C` — Is NewC = C? Continue if not
- `refute(NewC)` — Recurse with input NewC

# Conclusion
## Overview

With the Resolution method...

- we can find out if a formula is *valid* if we can show that its negation has a contradiction.
- we also can find out if a formula itself has a contradiction and thus is *unsatisfiable*.

# Conclusion
## Computational Complexity

- Problems: Pigeon Hole Comparison
- co-NP-complete problem

→ It is believed that there is no efficient algorithm for determining propositional validity.