Implementing an Inference Procedure

We've discussed rules of inference for propositional logic.

It would be useful from a computational point of view if we had an inference procedure that carried out more simply – say, in a single operation – the variety of processes involved in reasoning with the inference rules given.

Fortunately, there is such a procedure: **Resolution**

Resolution

Recall the following rules of inference:

Unit resolution

$$\frac{\alpha \vee \beta,\ \neg \beta}{\alpha}$$

Resolution

$$\frac{\alpha \vee \beta,\ \neg \beta \vee \gamma}{\alpha \vee \gamma}$$

We will base an inference procedure on the application of these rules of inference – ignoring the other rules.

Before we can do so, however, we have to be certain that the sentences (axioms) in our knowledge base are expressed in a form to which these rules can be applied: **clause form**.

Clause Form

A sentence in clause form is one

- Without $\wedge$
- Without $\Rightarrow$
- In which negation applies to single terms only

For example,

$(a \vee b)$
$(c \vee d \vee e)$
$(f \vee \neg g)$

Converting to Clause Form

In order to convert a sentence in propositional logic to clause form, one can follow these steps:

- Convert $a \Rightarrow b$ to $(\neg a \vee b)$

- Apply deMorgan's Laws so that any $\neg$ refers only to a single term:
  $\neg (a \wedge b) = (\neg a \vee \neg b)$
  $\neg (a \vee b) = (\neg a \wedge \neg b)$
  $\neg \neg a = a$

- Apply distributive law to convert to conjunctive normal form (i.e., a conjunction of disjuctions)
  $(a \wedge b) \vee c = (a \vee c) \wedge (b \vee c)$

- Make a separate clause for each conjunct.
  $(a \vee c)$
  $(b \vee c)$

Example. Converting the Work/Sleep Knowledge Base (KB)

Sun ∨ Mon ∨ Tues ∨ Wed ∨ Thurs ⇒ Work
¬ (Sun ∨ Mon ∨ Tues ∨ Wed ∨ Thurs) ∨ Work
(¬Sun ∧ ¬Mon ∧ ¬Tues ∧ ¬Wed ∧ ¬Thurs) ∨ Work
(¬Sun ∨ Work) ∧ (¬Mon ∨ Work) ∧ (¬Tues ∨ Work) ∧ (¬Wed ∨ Work) ∧ (¬Thurs ∨ Work)
(¬Sun ∨ Work)
(¬Mon ∨ Work)
etc.

Party ∧ Work ⇒ ¬Sleep
¬ (Party ∧ Work) ∨ ¬Sleep
(¬Party ∨ ¬Work) ∨ ¬Sleep
¬Party ∨ ¬Work ∨ ¬Sleep

Resolution: Proof by Refutation (Contradiction)

To prove that a sentence S is true, we will assume the opposite, and show that that leads to a contradiction with the knowledge base.

High-level view of the algorithm:

1. Negate S and convert the result to clause form. Add it to the KB.
2. Repeat until either a contradiction is found or no progress can be made:

- Select two clauses. Call these the parent clauses.
- Resolve the parent clauses. Call the resulting clause the resolvent.
- If the resolvent is empty, then a contradiction has been found. If it is not, then add it to the KB.

Example. Applying resolution to the Work/Sleep problem.

Our set of axioms (i.e., our knowledge base) is:
(¬Sun ∨ Work)
(¬Mon ∨ Work)
(¬Tues ∨ Work)
(¬Wed ∨ Work)
(¬Thurs ∨ Work)
(¬Thurs ∨ Party)
(¬Fri ∨ Party)
(¬Sat ∨ Party)
¬Party ∨ ¬Work ∨ ¬Sleep
Thurs

To prove ¬Sleep, we add Sleep to the KB.

<u>Thurs, (¬Thurs ∨ Work)</u>
Work　　　　　　　　add Work to KB

<u>Thurs, (¬Thurs ∨ Party)</u>
Party　　　　　　　　add Party to KB

<u>Work, ¬Party ∨ ¬Work ∨ ¬Sleep</u>
¬Party ∨ ¬Sleep　　　　add ¬Party ∨ ¬Sleep to KB

<u>Party, ¬Party ∨ ¬Sleep</u>
¬Sleep　　　　　　　　add ¬Sleep to KB

¬Sleep, Sleep　CONTRADICTION

Completeness of Resolution Proof by Contradiction

The algorithm given above is **complete**.

On the other hand, if we applied resolution in a "forward" direction (i.e, if we did not do a proof by contradiction), it would often work – but would not be complete!

Consider beginning with an empty KB. Say you want to prove $P \vee \neg P$
You can do this with a resolution proof by contradiction. But you cannot do it in a "forward" manner because there is nothing with which to resolve anything.

Is there anything faster?

Yes – if we **restrict** the expressiveness of our language.