RESOLUTION AND LOGIC-BASED AGENTS

Proof as Search

- Proof problems can easily be formulated as *search*, in the way that we formulated other problems.
- Suppose we want to establish whether $\phi_1, \ldots, \phi_n \vdash \psi$.
- State space: sequence of formulae.
- Initial state: $\phi_1, \ldots, \phi_n$.
- Goal: sequence of formulae with last element ψ.
- Operators: rules, which when applied to some elements in sequence generate new formula appended to state.

- Problems:
  - no solution guaranteed — perhaps non-terminating;
  - no way of knowing "right" rule to apply.
- Huge amounts of work on *heuristics for proof*.
- Small sets of sound & complete rules better …
- *resolution*
  - sound & complete proof method with just 1 rule.

Resolution

- Based on *checking satisfiability* of formulae.
- Relies on the fact that

$$\phi_1, \ldots, \phi_n \vdash \psi$$

iff

$$\phi_1 \wedge \cdots \wedge \phi_n \wedge \neg\psi$$

is unsatisfiable.

- So, negate what you want to show, add it to what you know, and try to show unsatisfiability.

- The resolution rule itself is:

$$\frac{\vdash \phi \lor \psi \qquad \vdash \chi \lor \neg\psi}{\vdash \phi \lor \chi} \text{ resolution}$$

- Why does this work? Reasoning by cases.
- Unsatisfiability is proved when we there is nothing left after we resolve two formulae together.

- The problem is that resolution only applies to disjunctions:

$$p \lor q \lor r \lor s \lor \ldots$$

- So we can't apply the rule to arbitrary formulae.
- . . . at least not without rewriting.
- It turns out that we can rewrite *any* formula in a suitable way.
- We can rewrite it as a conjunction of disjunctions of literals:

conjunctive normal form

- Consider how to do this on:

$$\neg(\phi \Rightarrow \psi) \lor (\chi \Rightarrow \phi)$$

1. Eliminate $\Rightarrow$:

$$\neg(\neg\phi \lor \psi) \lor (\neg\chi \lor \phi)$$

2. Move negation inwards:

$$(\phi \land \neg\psi) \lor (\neg\chi \lor \phi)$$

Using De Morgan and eliminating $\neg\neg$

3. Turn into a conjunction of disjunctions:

$$(\phi \lor \neg\chi \lor \phi) \land (\neg\psi \lor \neg\chi \lor \phi)$$

then

$$(\phi \lor \neg\chi) \land (\neg\psi \lor \neg\chi \lor \phi)$$

Using distribution laws.

- The final output is then a set of disjunctions:

$$\phi \lor \neg\chi, \neg\psi \lor \neg\chi \lor \phi$$

- So, the "given" set of formulae, which are implicitly a conjunction, are expanded into a bigger set, all of which are disjunctions.
- We can then use resolution on this set.
- Because resolution is not complete for constructive proofs ($\phi \land \psi \models \phi \lor \psi$), we can't proceed directly.
- But we can do proof by contradiction:
  1. negate the thing we want to show.
  2. resolve until we get the formula _, the empty clause.

• Used like this the process is:

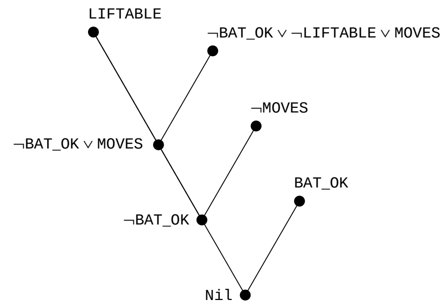
  – complete — if the goal is provable, the empty clause will be produced.

  – decidable — if the goal is not provable, the process will terminate without producing the empty clause.

• Consider doing this for:

$$
\begin{aligned}
BAT\_OK \\
\neg MOVES \\
\neg BAT\_OK \vee \neg LIFTABLE \vee MOVES
\end{aligned}
$$

• The resolution can then proceed like:



• How do we choose which formulae to resolve?

• Simplest idea is *breadth-first*—resolve everything with everything.

• This works, but like all breadth-first methods generates a *huge* amount of formulae.

• Also have *unit preference*—at least one resolvent is just one literal.

• *linear input*—at least one resolvent is one of the original clauses. (not complete)

• *set of support*—at least one resolvent is a descendent of the goal.

• (Note that the example uses unit preference and set of support.)

• This gives us the machinery of resolution for propositional logic.

• What about predicate logic?

• Well, we need to take care of quantifiers and variables.

• We deal with quantifiers in the following way.

• First, we *standardize* variables. To do this we give each quantifier its own variable.

$$\forall x \cdot \phi(x) \vee \exists x \cdot \psi(x)$$

becomes

$$\forall x \cdot \phi(x) \vee \exists y \cdot \psi(y)$$

• We can then eliminate existential quantifiers.

- We do this, as we did in the proof theory we looked at before, by using Skolemisation.
- Existentially quantified variables not in the scope of universal quantifiers can be replaced by Skolem constants.

  $$\exists x \cdot \phi(x)$$

  becomes:

  $$\phi(a)$$

  where a is a new constant symbol.

- Existentially quantified variables in the scope of universally quantifed variables can be replaced by Skolem functions of that variable.

  $$\forall y \cdot \psi(y) \Rightarrow \exists x \cdot \phi(x, y)$$

  becomes:

  $$\forall y \cdot \psi(y) \Rightarrow \phi(f(y), y)$$

- If ψ is "human" and ϕ is "mother", then $f(y)$ is the function that names everyone's mother.

- Once we have eliminated all the existential quantifers, it is easy to convert every formula into *prenex form*.
- We simply move all the universal quantfiers to the start of the formula.
- We can then get rid of the universal quantifiers.
- Every formula is implicitly universally quantified (or if it had no universal quantifiers it has no variables in it anymore).
- This gets us to a position in which we can start resolution.
- But how do we handle variables when we come to resolve?

- Clearly we can resolve:

  $$\phi(x) \vee \chi(x)$$

  and

  $$\psi(y) \vee \neg\phi(y)$$

  to get

  $$\psi(x) \vee \chi(x)$$

- But, what if the second formula was:

  $$\psi(b) \vee \neg\phi(c)$$

  ?

- In this case it is easy.
- To resolve we have to make the $\phi(x)$ and the $\neg\phi(c)$ match.
- We do this by instantiating x to have the value c.
- That gives the resolvent $\psi(b) \vee \chi(c)$
- More generally we have to identify the *most general unifier* of any variables in the two literals being eliminated, and use these to instantiate the remaining variables in the relevant clauses.
- However, we will not go into the detail of how to do unification.

Logic-Based Agents

- When we started talking about logic, it was as a means of representing knowledge.
- We wanted to represent knowledge in order to be able to build agents.
- We now know enough about logic to do that.
- We will now see how a *logic-based agent* can be designed to perform simple tasks.
- Assume each agent has a *database*, i.e., set of FOL-formulae.

  These represent information the agent has about environment.

- We'll write Δ for this database.
- Also assume agent has set of *rules*.

  We'll write R for this set of rules.
- We write $\Delta \vdash_R \phi$ if the formula ϕ can be proved from the database Δ using only the rules R.
- How to program an agent:

  Write the agent's rules R so that it should do action a whenever $\Delta \vdash_R Do(a)$.

  Here, Do is a predicate.
- Also assume A is set of actions agent can perform.

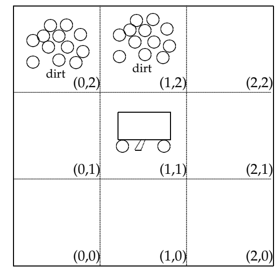
- The agent's operation:

```
for each a in A do
    if Δ ⊢_R Do(a) then
        return a
    end-if
end-for
for each a in A do
    if Δ ⊬_R ¬Do(a) then
        return a
    end-if
end-for
return null
```

- An example:

  We have a small robot that will clean up a house. The robot has sensor to tell it whether it is over any dirt, and a vacuum that can be used to suck up dirt. Robot always has an orientation (one of n, s, e, or w). Robot can move forward one "step" or turn right $90°$. The agent moves around a room, which is divided grid-like into a number of equally sized squares. Assume that the room is a 3×3 grid, and agent starts in square $(0, 0)$ facing north.

- Illustrated:



- Three *domain predicates* in this exercise:

$In(x, y)$	agent is at (x, y)
$Dirt(x, y)$	there is dirt at (x, y)
$Facing(d)$	the agent is facing direction d

- For convenience, we write rules as:

$$\phi(\ldots) \longrightarrow \psi(\ldots)$$

- First rule deals with the basic cleaning action of the agent

$$In(x, y) \wedge Dirt(x, y) \longrightarrow Do(suck) \tag{1}$$

- Hardwire the basic navigation algorithm, so that the robot will always move from $(0, 0)$ to $(0, 1)$ to $(0, 2)$ then to $(1, 2)$, $(1, 1)$ and so on.

- Once agent reaches $(2, 2)$, it must head back to $(0, 0)$.

$$\begin{aligned} In(0,0) \wedge Facing(north) \wedge \neg Dirt(0,0) &\longrightarrow Do(forward) & (2)\\ In(0,1) \wedge Facing(north) \wedge \neg Dirt(0,1) &\longrightarrow Do(forward) & (3)\\ In(0,2) \wedge Facing(north) \wedge \neg Dirt(0,2) &\longrightarrow Do(turn) & (4)\\ In(0,2) \wedge Facing(east) &\longrightarrow Do(forward) & (5) \end{aligned}$$

- Other considerations:
  - *adding* new information after each move/action;
  - *removing* old information.
- Suppose we scale up to 10×10 grid?

Summary

- This lecture covered two logic-related topics.
- First is covered mechanical theorem proving:
  - Pointed out some problems.
  - Suggested resolution as a solution.
- Next we looked at how logic might be used to program an agent.
  - Assumes we have a theorem prover.