EECS 391: Introduction to AI

Soumya Ray

Website: http://engr.case.edu/ray_soumya/eecs391_sp15/
Email: sray@case.edu
Office: Olin 516
Office hours: F 11:30-2pm (email if stopping by between 12:30-2) or by appointment
Today

• Converting FOL formulae to CNF
• Automated Planning (Ch 10)
Resolution Refutation Algorithm

• Convert \((KB \land \lnot \alpha)\) to universally-quantified CNF

• Starting with this KB, generate all possible consequences using resolution as operator

• Continue until:
  • No new clauses are generated. Then KB does NOT entail \(\alpha\)
  • Two clauses resolve to yield the empty clause. Then KB entails \(\alpha\)
CNF

- Universally quantified conjunctions over clauses
  - Clause: Disjunction of literals
Conversion to CNF

1. Standardize Apart:
   \[ \forall x P(x) \land \exists x Q(x) \rightarrow \forall x P(x) \land \exists y Q(y) \]

2. Eliminate implications:
   \[ \alpha \Rightarrow \beta \equiv \neg \alpha \lor \beta \]

3. Move negation inwards: de Morgan’s Laws,
   \[ \neg \forall x \ p \equiv \exists x \ \neg p, \ \neg \exists x \ p \equiv \forall x \ \neg p \]

4. Skolemize (get rid of \( \exists \))

5. “Drop” universal quantifiers

6. Distribute \( \land \) over \( \lor \)
Skolemization (remove $\exists$)

- If our KB has $\exists x P(x)$, we can substitute $x$ with some object from our model’s domain
  - We don’t specifically know the name of this object, so we will invent one
  - This is called a “Skolem constant”
- So $\exists x P(x)$ becomes $P(\text{NewSymbol})$, where \text{NewSymbol} is a constant symbol not used by anything else in our KB
Skolemization part 2

• If our KB has $\forall x \exists y P(x, y)$, the new object for $y$ might need to depend on $x$

• Example:

$$\forall x \exists y \text{Friend}(x, y) \models \forall x \text{Friend}(x, \text{Jeff})?$$

• In this case we use a “Skolem function”

$$\forall x \exists y \text{Friend}(x, y) \text{ becomes }$$
$$\forall x \text{Friend}(x, \text{FriendOf}(x))$$
Skolemization

- $\forall a,b,c,d... \exists y P(y) \rightarrow \forall a,b,c,d... P(f_{new}(a,b,c,d...))$
Example

\[ \forall x (\forall y \text{Animal}(y) \Rightarrow Loves(x, y)) \Rightarrow (\exists y \text{Loves}(y, x)) \]

1. \[ \forall x (\forall y \text{Animal}(y) \Rightarrow Loves(x, y)) \Rightarrow (\exists p \text{Loves}(p, x)) \]

2. \[ \forall x (\neg \forall y \neg \text{Animal}(y) \lor Loves(x, y)) \lor (\exists p \text{Loves}(p, x)) \]

3. \[ \forall x (\exists y \neg (\neg \text{Animal}(y) \lor Loves(x, y))) \lor (\exists p \text{Loves}(p, x)) \]

4. \[ \forall x (\forall y \text{Animal}(y) \land \neg Loves(x, y)) \lor (\exists p \text{Loves}(p, x)) \]

5. \[ \forall x (\text{Animal}(f(x)) \land \neg Loves(x, f(x))) \lor (\text{Loves}(g(x), x)) \]

6. \[ (\text{Animal}(f(x)) \lor Loves(g(x), x)), (\neg \text{Loves}(x, f(x)) \lor \text{Loves}(g(x), x)) \]
Example

$\forall x \text{ Feathers}(x) \land \text{Flies}(x) \Rightarrow \text{Bird}(x)$

$\forall x \text{ Bird}(x) \Rightarrow \text{Animal}(x)$

$\forall x \forall y \text{Animal}(x) \land \text{CanTalk}(x) \land \text{Human}(y) \Rightarrow \text{Likes}(x, y)$

$\text{Feathers(Parrot)}, \text{Flies(Parrot)}, \text{CanTalk(Parrot)}, \text{Human(Socrates)}$

$\neg \text{Likes(Parrot, Socrates)}$

\[
\neg \text{Feathers}(x) \lor \neg \text{Flies}(x) \lor \text{Bird}(x)
\]

\[
\neg \text{Bird}(x) \lor \text{Animal}(x)
\]

\[
\neg \text{Animal}(x) \lor \neg \text{CanTalk}(x) \lor \neg \text{Human}(y) \lor \text{Likes}(x, y)
\]

$\text{Feathers(Parrot)}, \text{Flies(Parrot)}, \text{CanTalk(Parrot)}, \text{Human(Socrates)}$

$\neg \text{Likes(Parrot, Socrates)}$
Example

\[\neg \text{Animal}(x) \lor \neg \text{CanTalk}(x) \lor \neg \text{Human}(y) \lor \text{Likes}(x, y), \neg \text{Likes(Parrot, Socrates)} \]

\[\neg \text{Animal(Parrot)} \lor \neg \text{CanTalk(Parrot)} \lor \neg \text{Human(Socrates)} \]

\[\neg \text{Animal(Parrot)} \lor \neg \text{CanTalk(Parrot)} \lor \neg \text{Human(Socrates)}, \text{Human(Socrates)} \]

\[\neg \text{Animal(Parrot)} \lor \neg \text{CanTalk(Parrot)} \]

\[\neg \text{Animal(Parrot)} \lor \neg \text{CanTalk(Parrot)}, \text{CanTalk(Parrot)} \]

\[\neg \text{Animal(Parrot)} \]

\[\neg \text{Bird}(x) \lor \text{Animal}(x), \neg \text{Animal(Parrot)} \]

\[\neg \text{Bird(Parrot)} \]

\[\neg \text{Feathers}(x) \lor \neg \text{Flies}(x) \lor \text{Bird}(x), \neg \text{Bird(Parrot)} \]

\[\neg \text{Feathers(Parrot)} \lor \neg \text{Flies(Parrot)} \]

\[\neg \text{Feathers(Parrot)} \lor \neg \text{Flies(Parrot)}, \text{Feathers(Parrot)}, \text{Flies(Parrot)} \]

\[\square\]
Variants of Resolution

• In order to make the resolution procedure more efficient, people have explored several variants
• These are heuristics that (in some cases) sacrifice completeness for efficiency
• Unit Resolution
• Linear Resolution
• Book has others
Summary

• We learned about:
  – FOL syntax and semantics
  – Lifting and Unification
  – Resolution and the semidecidability of FOL inference
  – Conversion to CNF
Automated Planning (Ch 10)

• Consider a situation where an agent has to carry out a sequence of actions to achieve a goal

• Suppose the agent starts off with detailed, *structured* knowledge of the world
  – Could we take advantage of this?
  – E.g. a chess playing agent should start knowing rules
The Planning Problem

• Given:
  – An initial state of the world, described as a set of logical facts
  – A set of goal states, described as a set of logical facts
  – A set of actions, also described in logic

• Find a sequence of actions that will move the world from the initial state to the final state
  – This sequence is called a *plan*
  – Often also try to optimize some criteria
“Classical” Planning

• We’ll study planning algorithms designed to work when the world is:
  – Deterministic
  – Static
  – Fully observable
  – Propositional
  – Actions are instantaneous

• These restrictions can be relaxed (more or less)
Blocks World

Task: Starting with initial configuration of blocks, produce a desired goal configuration by moving block around.
Situation Calculus (Chapter 10.3)

• It is natural to think of using full FOL to encode states of the world and actions
  – Then use general FOL inference as planner

• People developed a general method for encoding states and actions
  – Called the Situation Calculus
Issues with Situation Calculus

• SC is appealing because no special algorithms are needed for planning
  – Given an SC knowledge base, query “Is there a sequence of actions leading to a situation where the goal holds?”
  – Apply resolution
• But this is very slow, even for small planning problems
• So specialized fragments of FOL have been developed to represent planning problems instead
Representing a Planning Problem

• For classical planning, one fragment of FOL that is used is called STRIPS ("Stanford Research Institute Problem Solver")

• States, actions and goals will be represented in this language
  – Then we’ll see planning algorithms (which are inference algorithms in disguise) that find plans in this language
Representing States in STRIPS

- States in STRIPS are conjunctions of unnegated, ground, function-free literals
  - All conditions that hold in that state
  - $\text{Block}(A), \text{Block}(B), \text{On}(A,B), \text{On}(B, \text{Table}), \text{GripperEmpty}$
  - The “Closed World Assumption” is used
Representing Goals in STRIPS

• Goals are conjunctions of unnegated, ground, function-free literals

• Goals may not fully determine a state of the world
  – In this case, the goal is any state where these literals hold

• Example: $On(A,E) \land On(B,D)$
Representing Actions in STRIPS

• Want to represent an action of picking up a block from the table

\( \text{Pickup\_from\_Table}(x) \)

**Preconditions:** \( \text{Block}(x), \text{GripperEmpty}, \text{Clear}(x), \text{On}(x,\text{Table}) \)

**Add List:** \( \text{Holding}(x) \)

**Delete List:** \( \neg \text{GripperEmpty}, \neg \text{On}(x,\text{Table}) \)

“Applicability”: action can be used at a state iff its preconditions are satisfied
Representing Actions in STRIPS

• An “action schema” represents a non-ground action using three parts:
  – The action name and parameter list
  – The **preconditions**: a list of unnegated function-free (non-ground) literals. Any variables in this list are parameters to the action.
  – The **effects**: a list of function-free literals describing how the state changes.
Add and Delete Lists

• Often, the unnegated literals in the action effects are collected into an “ADD” list, and the negated literals are collected into a “DELETE” list
  – Idea: Starting with initial state, to get result of applying action, add the literals in ADD list and delete the literals in the DELETE list
The STRIPS assumption

• Every possible effect of actions are listed
  – i.e., if a literal does not appear in the effects list, it is unchanged in the resulting state
  – Solves the “frame problem” in situation calculus
Restrictions in STRIPS

• States are described by unnegated ground function-free literals
• CWA
• Ground conjunctive goals
• Conjunctive effects
• No equality
Example: Blocks World

B  C  A

Table
Example

- **Init**($On(A, Table) \land On(B, Table) \land On(C, Table) \land Block(A) \land Block(B) \land Block(C) \land Clear(A) \land Clear(B) \land Clear(C)$)
- **Goal**($On(A, B)$)
- **Action**($MoveToTable(b, x)$,
  - **Preconditions**($On(b, x) \land Clear(b) \land Block(b) \land Block(x)$)
  - **AddEffects**($On(b, Table) \land Clear(x)$)
  - **DelEffects**($On(b, x)$)
Planning Algorithms

• Given a STRIPS representation of a classical planning problem, how do we solve it?
  – Since the world is static, deterministic, fully observable, we could use search
  – Remember that in this case, the search algorithm is actually performing logical inference
Kinds of Search

• Search algorithms for classical planning fall into two categories
  – “State space planners”: States of the search problem are states of the world; search operators are actions of the world
  – “Plan space planners”: States of the search problem are partial plans; search operators are modifications to the current partial plan
Forward State-Space Search

• “Progression” planning

• Setup:
  – States=world states (in STRIPS)
  – Initial state=given
  – Operators=applicable actions (in STRIPS)
  – Goal test=given (in STRIPS)
  – Operator costs=unit (minimize number of actions)
Forward State-Space Search

• We could apply any search algorithm, e.g. A*

• The key differences are:
  – Only applicable actions need to be explored at a state
  – Getting the next state is done through the STRIPS specification of states and actions
  – Heuristics are based on planning ideas
Search Heuristics

• From any state, want to estimate the number of actions to search termination admissibly

• Two possibilities:
  – Relax the planning problem
  – Consider subproblems
Relaxed Plans

• There are different ways to arrive at a less constrained planning problem

• One way is to remove all DELETE effects from STRIPS actions
  – This is admissible (why?)
  – To estimate this cost, need to run an internal planning loop; but this is usually very fast
Subproblems

• The goal is a conjunction of literals

• We can generate subproblems by just considering a single literal at a time
  — “Subgoal Independence” (admissible)

• Combine with max, as usual
Total Order Plans

• In a Total Order plan, every pair of actions $A_1$ and $A_2$ has a *temporal ordering constraint*
  
  – Either $A_1$ is done first, or $A_2$
  
  – Forward state space planners produce plans like this
Partial Order Plans

• In many situations, actions do not have to be done in order
  – Might be trying to achieve unrelated things

• This is a partial order plan: a plan with some actions that have no temporal ordering constraints between them
  – i.e. there is some \( A_1, A_2 \) so that \( A_1 \) does not have to be completed before \( A_2 \) and \( A_2 \) does not have to be completed before \( A_1 \) for the plan to succeed
Blocks World

Goal: $\text{On}(A,B) \land \text{On}(C,D)$

A dummy action with no preconditions and effect==initial state

Start

Move(A,Table,B)  Move(C,Table,D)

A dummy action with no effects and preconditions==goal

End
Partial Order Plans

• Represented as a set of actions and ordering constraints \((A < B)\)

• Partial Order plans have two advantages over total order plans
  – Flexibility when executing the plan
  – Action Parallelism
Finding a POP

• To find POPs, we will perform a *plan-space* search

• The states of the search space will be *incomplete POPs*, augmented with some bookkeeping information

• Starting with an empty POP (only “Start” and “End”), we will add actions and ordering constraints until we have a complete POP (or fail)
States of the Search Space

• A state will have
  – The incomplete POP (list of actions and ordering constraints)
    • Ordering constraints can’t introduce cycles
  – A list of open conditions (initially the goal)
    • Open condition: A literal that the plan needs to make true that is not currently the effect of some action in the plan
    • The termination states of the search are those where this list is empty
  – A list of “causal links”
Causal Links

- Suppose an action $B$ in the incomplete POP has a precondition $p$
- Suppose I add an action $A$ to the POP that has $p$ as an effect
- The cause for adding $A$ is so it fulfills $p$ for $B$
- This is indicated in the POP by adding a causal link, denoted $A \xrightarrow{p} B$
Conflict and Threats

• Suppose I have just added $A$ and a causal link $A \rightarrow B$

• Suppose there is an action $C$ in the plan that has $\neg p$ as an effect and could come after $A$ and before $B$

• We say that $C$ conflicts with, or threatens, the causal link $A \rightarrow B$
Consistent Plans

• A consistent plan is a POP where the ordering constraints have no cycles and there are no conflicts with causal links

• So a solution is a consistent plan with no open conditions
POP Algorithm

• Initial State: Plan=(\(Start, End, Start < End\)),
  open conditions=preconditions of \(End\) (goal),
  no causal links

• Search operators: Pick an open condition, and
  an action to satisfy it; generate next state
  (POP)

• Goal test: empty list of open conditions
Generating the Next State

• Suppose we pick condition \( p \) on action \( B \) to satisfy, using action \( A \)
  – \( A \) might be an action already in the plan

• To generate the next state:
  – Remove \( p \) from list of open conditions
  – Add \( A \) to list of actions in POP
  – Add ordering constraints \( Start < A, A < B, A < End \)
  – Add \( A \)'s preconditions to list of open conditions
  – Add a causal link \( A \rightarrow B \)
  – Resolve conflicts/threats if any
Conflict Resolution

• Conflicts can arise between
  – The new causal link and existing actions
  – The new action and existing causal links

• To resolve a conflict, add ordering constraints
  – Suppose $C$ conflicts with $A \rightarrow B$
  – Then add either $C < A$ or $B < C$ (assuming no cycles)
  – This is a branch point in the search
Example

- **Init**: $At(Flat,Axle), At(Spare,Trunk)$
- **Goal**: $At(Spare,Axle)$
- **Remove**(Spare,Trunk)
  - **PRE**: $At(Spare,Trunk)$
  - **EFF**: $\neg At(Spare,Trunk), At(Spare,Ground)$
- **Remove**(Flat,Axle)
  - **PRE**: $At(Flat,Axle)$
  - **EFF**: $\neg At(Flat,Axle), At(Flat,Ground)$
- **PutOn**(Spare,Axle)
  - **PRE**: $At(Spare, Ground), \neg At(Flat,Axle)$
  - **EFF**: $\neg At(Spare, Ground), At(Spare,Axle)$
- **LeaveOvernight**
  - **PRE**:
  - **EFF**: $\neg At(Spare, Trunk), \neg At(Spare, Ground), \neg At(Spare,Axle), \neg At(Flat,Axle), \neg At(Flat,Ground)$
Example

Start
\[\text{At(Flat,Axle)}, \, \text{At(Spare,Trunk)}\]

\[\text{At(Spare,Trunk)}\]

Remove(Spare,Trunk)
\[\neg \text{At(Spare,Trunk)}, \, \text{At(Spare,Ground)}\]

\[\text{At(Spare,Ground)}\]

\[- \text{At(Flat,Axle)}\]

\[- \text{At(Spare,Ground)}\]

\[- \text{At(Spare,Trunk)}\]

PutOn(Spare,Axle)

\[- \text{At(Spare,Ground)}\]

\[- \text{At(Spare,Axle)}\]

\[\text{At(Spare,Axle)}\]

End

LeaveOvernight
\[\neg \text{At(Flat,Axle)}, \, \neg \text{At(Spare,Ground)}, \, \neg \text{At(Spare,Trunk)}\]...
Example

Start
\( At(Flat, Axle), At(Spare, Trunk) \)

\( At(Spare, Trunk) \)
Remove(Spare,Trunk)
\( \neg At(Spare, Trunk), At(Spare, Ground) \)

\( At(Flat, Axle) \)
Remove(Flat,Axle)
\( \neg At(Flat, Axle), At(Flat, Ground) \)

\( At(Spare, Ground), \neg At(Flat, Axle) \)
PutOn(Spare,Axle)
\( \neg At(Spare, Ground), At(Spare, Axle) \)

\( At(Spare, Axle) \)
End
Case Study: Deep Space 1
(Active mission Oct 24 1998-Dec 18 2001)
Deep Space 1 Technologies

• Tested three key techniques:
  – Ion propulsion system
  – Automatic intelligent navigation
  – Automatic planning to handle low-level actions and failure recovery
    • Nonclassical planning, need temporal actions and resource aware planners
Controller Architecture

Remote Agent (RA)

Mission Manager

Plan Execution

Fault Diagnosis and Recovery

Experiment Manager

Real time Flying Software

Hardware

Special Services

Human commands
Planner/Scheduler

- Mission Manager
- Special Services
- Domain Knowledge
- Search Heuristics
- Plan Search Engine
- Heuristic Scheduling
- Temporal Database
- Multithreaded Plan Execution
Summary

• We learned about:
  — Classical Planning
  — Forward State Space search
  — Partial Order Planning