# EECS 391: Introduction to AI

Soumya Ray

Website: http://vorlon.case.edu/~sray/eecs391_sp12/index.html

Email: sray@case.edu

Office: Olin 516

Office hours: Tue 2:30-4:00 or by appointment

# Announcements

- HW4 online today
- PA2 due
- Feng's office hours this week F 2:30-4pm

# Today

- FOL Inference (Chapter 9)
- Automated Planning (Chapter 10.1-2, 10.4.2, 10.4.4)

# Resolution inference rule

$$\frac{l_1 \vee l_2 \vee m \ldots \vee l_k, \quad r_1 \vee r_2 \vee \neg m \ldots \vee r_k}{l_1 \vee l_2 \ldots \vee l_k \vee r_1 \vee r_2 \ldots \vee r_k}$$

$$\frac{l_1 \vee l_2 \vee m_i \ldots \vee l_k, \quad r_1 \vee r_2 \vee \neg n_j \ldots \vee r_k, m_i\theta = n_j\theta}{l_1 \vee l_2 \ldots \vee l_k \vee r_1 \vee r_2 \ldots \vee r_k \{\theta\}}$$

Atomic Formulae

Unification: This substitution makes $m_i$ syntactically identical to $n_j$

# Unification Examples

$$Unify(P(A,x),P(A,C)) = \{x/C\}$$

$$Unify(P(A,x),P(y,B)) = \{x/B, y/A\}$$

$$Unify(P(A,x),Q(m,n)) = fail$$

$$Unify(P(v,x),P(y,f(u))) = \{x/f(u), v/y\}$$

# Most General Unifier (MGU)

- In general, multiple unifiers exist for two formulae
- E.g. $p(x,y)$ and $p(m,A)$ can be unified with $\{x/A,m/A,y/A\}$, $\{x/B,m/B,y/A\}$,..., $\{x/m,y/A\}$
- A substitution $\theta_1$ is *more general than* $\theta_2$ if there is a nontrivial substitution $\sigma$ so that $\theta_1\sigma=\theta_2$

# “More General Than” Example

- $\theta_1=\{x/m, y/A\}$, $\theta_2=\{x/A, m/A, y/A\}$
  - $\sigma=\{m/A\}$, $\theta_1\sigma=\{x/A, m/A, y/A\}= \theta_2$

- $\theta_1=\{x/m, y/A\}$, $\theta_2=\{x/B, m/B, y/A\}$
  - $\sigma=\{m/B\}$, $\theta_1\sigma=\{x/B, m/B, y/A\}= \theta_2$

# Most General Unifier (MGU)

- For any two unifiable formulae, there is a single MGU (upto renaming)

- So if we can find this MGU, we can use Resolution for FOL inference

# Unification Algorithm

- Input: Two formulae $s_1$ and $s_2$, standardized apart
- Start with a list containing "$s_1 = s_2$"
- Repeat until none of the following applies:
  - Three cases: $x=x$ (Case 1, remove) or $x=T$ or $T_1=T_2$
  - Case 2: Select $x=T$ or $T=x$ where $x$ is a variable and $T$ is a term and $x$ occurs in somewhere else in the list
    - If $x$ occurs in $T$, FAIL ("occurs check")
    - Else apply substitution $\{x/T\}$ to all other elements of list
  - Case 3: Select "$T_1=T_2$" on the list, where neither is a variable
    - $T_1$, $T_2$ constants? If the same, remove, else FAIL
    - $T_1$ is $g(E_1,\ldots,E_n)$ and $T_2$ is $g(F_1,\ldots,F_n)$? Erase $T_1=T_2$ and replace with $E_1=F_1,\ldots,E_n=F_n$
    - Else FAIL
- At end, return $\{x_i/T_i\}$, the elements left on the list

# “Occurs Check”

- Suppose we want to unify $p(x)$ and $p(f(x))$ (after standardizing apart)

- This isn’t possible
  - $\{x/f(x)\}$ does not work
  - Leads to infinite loop

# Example

$$Unify(p(f(x), y, x), p(a, a, f(C)))$$

$$\{p(f(x), y, x) = p(a, a, f(C))\}$$

$$\{a = f(x), y = a, x = f(C)\}$$

$$\{a = f(f(C)), y = a, x = f(C)\}$$

$$\{a = f(f(C)), y = f(f(C)), x = f(C)\}$$

$$p(f(f(C)), f(f(C)), f(C))$$

# Unification Algorithm

- Input: Two formulae $s_1$ and $s_2$, standardized apart
- Start with a list containing "$s_1 = s_2$"
- Repeat until none of the following applies:
  - Three cases: $x=x$ (Case 1, remove) or $x=T$ or $T_1=T_2$
  - Case 2: Select $x=T$ or $T=x$ where $x$ is a variable and $T$ is a term and $x$ occurs in somewhere else in the list
    - If $x$ occurs in $T$, FAIL ("occurs check")
    - Else apply substitution $\{x/T\}$ to all other elements of list
  - Case 3: Select "$T_1=T_2$" on the list, where neither is a variable
    - $T_1$, $T_2$ constants? If the same, remove, else FAIL
    - $T_1$ is $g(E_1,\ldots,E_n)$ and $T_2$ is $g(F_1,\ldots,F_n)$? Erase $T_1=T_2$ and replace with $E_1=F_1,\ldots,E_n=F_n$
    - Else FAIL
- At end, return $\{x_i/T_i\}$, the elements left on the list

# Algorithm

- Same as in propositional logic. We can use resolution in a search algorithm as follows:
  - Convert (KB $\wedge \neg\alpha$) to 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 $\alpha$

# Conversion to CNF

1. Eliminate implications: $\alpha \Rightarrow \beta \equiv \neg\alpha \vee \beta$
2. Move negation inwards: de Morgan's Laws,
   $\neg\forall x\ p \equiv \exists x\ \neg p, \quad \neg\exists x\ p \equiv \forall x\ \neg p$
3. Rename variables:
   $$\forall x P(x) \wedge \exists x Q(x) \rightarrow \forall x P(x) \wedge \exists y Q(y)$$
4. Skolemize: $\forall x \exists y P(x, y) \rightarrow \forall x P(x, F(x))$
5. Drop universal quantifiers
6. Distribute $\wedge$ over $\vee$

"Skolem Function" arguments are all universally quantified variables in whose scope the $\exists$ appears

# Skolem Function Example

$$\forall x \exists y Friend(x, y)$$

$$\forall x Friend(x, G(x))$$

# Conversion to CNF Example

$$\forall x(\forall y Animal(y) \Rightarrow Loves(x, y)) \Rightarrow (\exists y Loves(y, x))$$

$$1.\forall x(\neg \forall y \neg Animal(y) \vee Loves(x, y)) \vee (\exists y Loves(y, x))$$

$$2.\forall x(\exists y \neg(\neg Animal(y) \vee Loves(x, y))) \vee (\exists y Loves(y, x))$$

$$\forall x(\exists y Animal(y) \wedge \neg Loves(x, y)) \vee (\exists y Loves(y, x))$$

$$3.\forall x(\exists y Animal(y) \wedge \neg Loves(x, y)) \vee (\exists p Loves(p, x))$$

$$4.\forall x(Animal(f(x)) \wedge \neg Loves(x, f(x))) \vee (Loves(g(x), x))$$

$$5.(Animal(f(x)) \wedge \neg Loves(x, f(x))) \vee (Loves(g(x), x))$$

$$6.(Animal(f(x)) \vee Loves(g(x), x)) \wedge (\neg Loves(x, f(x)) \vee Loves(g(x), x))$$

# Example

$\forall x\ Feathers(x) \wedge Flies(x) \Rightarrow Bird(x)$

$\forall x\ Bird(x) \Rightarrow Animal(x)$

$\forall x \forall y\ Animal(x) \wedge CanTalk(x) \wedge Human(y) \Rightarrow Likes(x, y)$

$Feathers(Parrot), Flies(Parrot), CanTalk(Parrot), Human(Socrates)$

$?\ Likes(Parrot, Socrates)$

---

$\neg Feathers(x) \vee \neg Flies(x) \vee Bird(x)$

$\neg Bird(x) \vee Animal(x)$

$\neg Animal(x) \vee \neg CanTalk(x) \vee \neg Human(y) \vee Likes(x, y)$

$Feathers(Parrot), Flies(Parrot), CanTalk(Parrot), Human(Socrates)$

$\neg Likes(Parrot, Socrates)$

# Example

$$\frac{\neg Animal(x) \vee \neg CanTalk(x) \vee \neg Human(y) \vee Likes(x, y), \neg Likes(Parrot, Socrates}{\neg Animal(Parrot) \vee \neg CanTalk(Parrot) \vee \neg Human(Socrates)}$$

$$\frac{\neg Animal(Parrot) \vee \neg CanTalk(Parrot) \vee \neg Human(Socrates), Human(Socrates)}{\neg Animal(Parrot) \vee \neg CanTalk(Parrot)}$$

$$\frac{\neg Animal(Parrot) \vee \neg CanTalk(Parrot), CanTalk(Parrot)}{\neg Animal(Parrot)}$$

$$\frac{\neg Bird(x) \vee Animal(x), \neg Animal(Parrot)}{\neg Bird(Parrot)}$$

$$\frac{\neg Feathers(x) \vee \neg Flies(x) \vee Bird(x), \neg Bird(Parrot)}{\neg Feathers(Parrot) \vee \neg Flies(Parrot)}$$

$$\frac{\neg Feathers(Parrot) \vee \neg Flies(Parrot), Feathers(Parrot), Flies(Parrot)}{\square}$$

# 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

# Automated Planning

- A form of “sequential decision making”

- 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 facts
  - A set of actions, also described in logic

- Find a short 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

# Situation Calculus

- It is natural to think of using full FOL to encode states of the world and actions
  - Then use general FOL inference as a planner
  - Called the “Situation Calculus”

# Situation Calculus

- A "*situation*" is a logical term that summarizes the current state of the world (state + time index)
- In each situation, the agent can take an action (another logical term), to get a new situation
- *Fluents* are functions or predicates that vary from one situation to the next

# Situation Calculus Examples

$At(Agent, [1,1], S_0)$

$\neg Holding(Agent, Gold, S_0)$

$At(Agent,x,s) \wedge Adjacent(x,y) \Rightarrow Poss(Go(x,y),s)$

$Result([],s)=s$

$Result([a,b],s)=Result(b,Result(a,s))$

$Poss(Go(x,y),s) \Rightarrow At(Agent,y,Result(Go(x,y),s))$

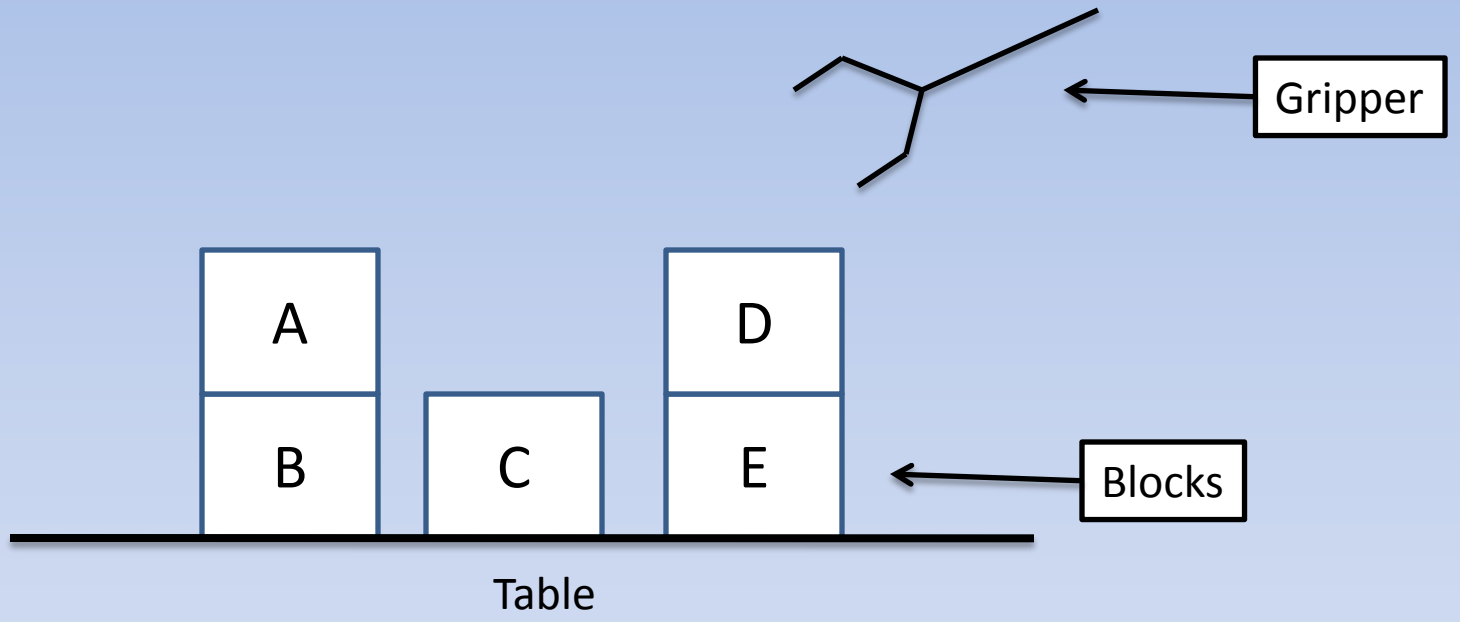
# Issues

- The 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 desired fluent 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

# “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.

# 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
  - *Block(A)*, *Block(B)*, *On(A,B)*, *On(B, Table)*, *GripperEmpty*
  - The “Closed World Assumption” is used

# Closed World Assumption

- If any literal is not explicitly mentioned in the state, it is taken to be false

- This prevents the description of states becoming too large

- In our example, no need to say $\neg On(A,C)$, $\neg On(A,D)$, $\neg On(A,E)$,…

# Representing Goals in STRIPS

- Goals are conjunctions of unnegated function-free ground literals
- Goals may not fully determine a state of the world
  - In this case, the goal is any state where these literals hold
- Example: $On(A,E) \wedge On(B,D)$

# Representing Actions in STRIPS

- Want to represent an action of picking up a block from the table
- *Pickup_from_Table(x)*
- Preconditions: *Block(x)*, *GripperEmpty*, *Clear(x)*, *On(x,Table)*
- Effects: ¬*GripperEmpty*, *Holding(x)*, ¬*On(x,Table)*

# 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 an action, *add* the literals in ADD list and *delete* the literals in the DELETE list