COMP 424 - Artificial Intelligence
Lecture 11: Planning

Instructor: Jackie CK Cheung (jcheung@cs.mcgill.ca)

Readings: R&N Ch 10

Plan for Today

- Logical reasoning:
  - Syntax and semantics
    - Propositional logic
    - First-order logic
  - Proofs and resolution: Truth tables, inference rules, resolution

Today

- Planning:
  - STRIPS notation for describing planning tasks.
  - State-space planning.

First-order logic (FOL)

- Add a few **new elements**:
  1. **Predicates** are used to describe objects, properties, relationships.
  2. **Quantifiers** (∀ = *"for all"*, ∃ = *"there exists"*) are used for statements that apply to a class of objects.
  3. **Functions** are used to give you an object that is related to another object in a specific way. (e.g., the cell to the RightOf cell 1,1 is cell 2,1)
     - These objects (**domain elements**) are drawn from the **domain**.

Example: FOL sentence

$\forall x$ On(x,Table) $\rightarrow$ Fruit(x)

$\forall$ is a **quantifier**

x is a **variable**

Table is a **constant**

On is a **predicate**

Note: Quantifiers allows FOL to handle **infinite domains**, while propositional local can only handle finite domains.

Syntax of FOL: Basic elements

- **Connectives** $\wedge$, $\vee$, $\neg$, $\Rightarrow$
- **Variables** *x, y, …*
  - Ranges over the domain
- **Quantifiers** $\forall$, $\exists$
- **Predicates** At(Wumpus,*x*,*y*), IsPit(*x*,*y*), …
  - Map domain element(s) to True/False
  - **Equality** = predicate that checks whether two objects refer to the same domain element
- **Functions** SonOf(*x*), PlusOne(*x*)
  - Map domain element(s) to domain element
  - **Constants** map 0 domain elements to a domain element

    Wumpus, 2, CS424, …

Unification

Pattern matching to find promising candidates for UE

We say a substitution σ unifies atomic sentences p and q if $p\sigma = q\sigma$.

e.g.

p	q	σ
$Knows(John, x)$	$Knows(John, Jane)$	$\{x/Jane\}$
$Knows(John, x)$	$Knows(y, Mary)$	$\{y/John, x/Mary\}$
$Knows(John, x)$	$Knows(y, Mother(y))$	$\{y/John, x/Mother(John)\}$

Assume free variables in table are universally quantified

"If we plug in John for y and Mother(John) for x, the two sides are equal."

Unification

Idea: Unify complex sentences with known facts to draw conclusions.

p	q	σ
$Knows(John, x)$	$Knows(John, Jane)$	$\{x/Jane\}$
$Knows(John, x)$	$Knows(y, Mary)$	$\{y/John, x/Mary\}$
$Knows(John, x)$	$Knows(y, Mother(y))$	$\{y/John, x/Mother(John)\}$

e.g., If we know everything in *q* and also:

Knows(John,x) $\rightarrow$ *Likes(John,x)*

Then we can conclude:

Likes(John, Jane)

Likes(John, Mary)

Likes(John, Mother(John))

Generalized Modus Ponens (GMP)

$$\frac{p_1',\ p_2',\ \ldots,\ p_n',\ (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{q\sigma}$$ where $p_i'\sigma = p_i\sigma$ for all i

E.g.

p_1'	= *Faster(Bob,Pat)*
p_2'	= *Faster(Pat,Steve)*
$p_1 \wedge p_2 \Rightarrow q$	= *Faster(x,y)* $\wedge$ *Faster(y,z) => Faster(x,z)*
σ	= *x/Bob, y/Pat, z/Steve*
$q\sigma$	= *Faster(Bob,Steve)*

Generalized Modus Ponens (GMP)

$$\frac{p_1',\ p_2',\ \ldots,\ p_n',\ \ (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{q\sigma}$$ where $p_i'\sigma = p_i\sigma$ for all i

E.g.

p_1'	= *Faster(Bob,Pat)*
p_2'	= *Faster(Pat,Steve)*
$p_1 \wedge p_2 \Rightarrow q$	= *Faster(x,y)* $\wedge$ *Faster(y,z) => Faster(x,z)*
σ	= *x/Bob, y/Pat, z/Steve*
$q\sigma$	= *Faster(Bob,Steve)*

GMP used with KB of **Horn clauses** (=exactly 1 positive literal):

- a single atomic sentence;
- or a clause of the form: (conjunction of atomic sentences) => (atomic sentence).

- All variables assumed to be universally quantified.

Completeness in FOL

- Procedure *i* is complete if and only if:

$$KB \vdash_i \alpha \quad \text{whenever} \quad KB \models \alpha$$

- GMP is **complete** for KBs of universally quantified Horn clauses, but incomplete for general first-order logic.
- Entailment in FOL is only **semi-decidable**: can find a proof when KB entails α, but not always when KB does not entail α.
  - Reduces to **Halting Problem**: proof may be about to terminate with success or failure, or may go on forever.

Inference algorithms for FOL

1. **Propositionalize** the FOL into propositional logic
   - Too expensive for all but the most trivial cases!
   - See R&N 9.1 for more details
2. **Search**
   - Forward/backward chaining using generalized modus ponens
3. **Resolution**

Resolution

Two clauses can be resolved if they contain complementary literals, where one literal unifies with the negation of the other

- Intuition: Like resolution in propositional logic, but taking into account possible unifications

Example:

$Animal(F(x)) \vee Loves(G(x), x)$ $\quad\quad$ $\neg Loves(u, v) \vee \neg Kills(u, v)$

resolves to

$Animal(F(x)) \vee \neg Kills(G(x), x)$

because

$Loves(G(x), x)$ unifies with $\neg Loves(u, v)$ using substitution $\theta = \{u/G(X), v/x\}$

Resolution

- Sound and complete inference method for FOL.
- **Proof by negation**: To prove that KB entails α, instead we prove that ($KB \wedge \neg\alpha$) is **unsatisfiable**.
- **Method**:
  - The KB and $\neg\alpha$ are expressed in universally quantified, **conjunctive normal form**.
  - Repeat: The resolution inference rule combines two clauses to make a new one.
  - Continue until an empty clause is derived (contradiction).

Conjunctive Normal Form in FOL

- Literal = (possibly negated) atomic sentence, e.g., *¬Rich(Me)*
- Clause = disjunction of literals, e.g. *¬Rich(Me) v Unhappy(Me)*
- The KB is a big conjunction of clauses.

E.g. *¬Rich(x) v Unhappy(x)*

Rich(Me)

Unhappy(Me)

with $\sigma = \{x / Me\}$

Converting a KB to CNF

1. Replace $P \Rightarrow Q$ by $\neg P \vee Q$
2. Move $\neg$ inwards, e.g. $\neg \forall x\ P$ becomes $\exists x\ \neg P$
3. Standardize variables apart, e.g. $\forall x\ P \vee \exists x\ Q$ becomes $\forall x\ P \vee \exists y\ Q$
4. Move quantifiers left in order, e.g. $\forall x\ P \vee \exists y\ Q$ becomes $\forall x \exists y\ P \vee Q$
5. Eliminate existential quantifiers by **Skolemization**

Skolemization

- We want to get ride of existentially quantified variables:

  $\exists x\ Rich(x)$ becomes $Rich(G1)$

  where $G1$ is a new **Skolem constant**.

- It gets more tricky when $\exists$ is inside $\forall$

e.g. "Everyone has a heart"

$$\forall x\ Person(x) \Rightarrow \exists y\ Heart(y) \wedge Has(x,y)$$

How should we replace y here?

- Incorrect: $\forall x\ Person(x) \Rightarrow Heart(H1) \wedge Has(x,H1)$
- Correct: $\forall x\ Person(x) \Rightarrow Heart(H(x)) \wedge Has(x,H(x))$

  where H is a new symbol called a **Skolem function**.

Converting a KB to CNF

1. Replace $P \Rightarrow Q$ by $\neg P \vee Q$
2. Move $\neg$ inwards, e.g. $\neg \forall x\ P$ becomes $\exists x\ \neg P$
3. Standardize variables apart, e.g. $\forall x\ P \vee \exists x\ Q$ becomes $\forall x\ P \vee \exists y\ Q$
4. Move quantifiers left in order, e.g. $\forall x\ P \vee \exists x\ Q$ becomes $\forall x \exists y\ P \vee Q$
5. Eliminate existential quantifiers by Skolemization
6. Drop universal quantifiers
7. Distribute over $\vee$, e.g. $(P \wedge Q) \vee R$ becomes $(P \vee R) \wedge (Q \vee R)$

R&N 9.5.1

Example

Jack owns a dog.

Every dog owner is an animal lover.

No animal lover kills an animal.

Either Jack or Curiosity killed the cat, who is named Tuna.

Did Curiosity kill the cat?

Sentences + background knowledge

1. $\exists x : Dog(x) \wedge Owns(Jack, x)$

2. $\forall x; \quad (\exists y \quad Dog(y) \wedge Owns(x, y)) \rightarrow AnimalLover(x)$

3. $\forall x; \quad AnimalLover(x) \rightarrow (\forall y \quad Animal(y) \rightarrow \neg Kills(x, y))$

4. $Kills(Jack, Tuna) \vee Kills(Curiosity, Tuna)$

5. $Cat(Tuna)$

6. $\forall x : \quad Cat(x) \rightarrow Animal(x)$

Example: Conjunctive Normal Form

$Dog(D)$

(D is a placeholder for the dogs unknown name (i.e. Skolem symbol/function). Think of D like "JohnDoe")

$Owns(Jack, D)$

$\neg Dog(y) \vee \neg Owns(x, y) \vee AnimalLover(x)$

$\neg AnimalLover(w) \vee \neg Animal(y) \vee \neg Kills(w, y)$

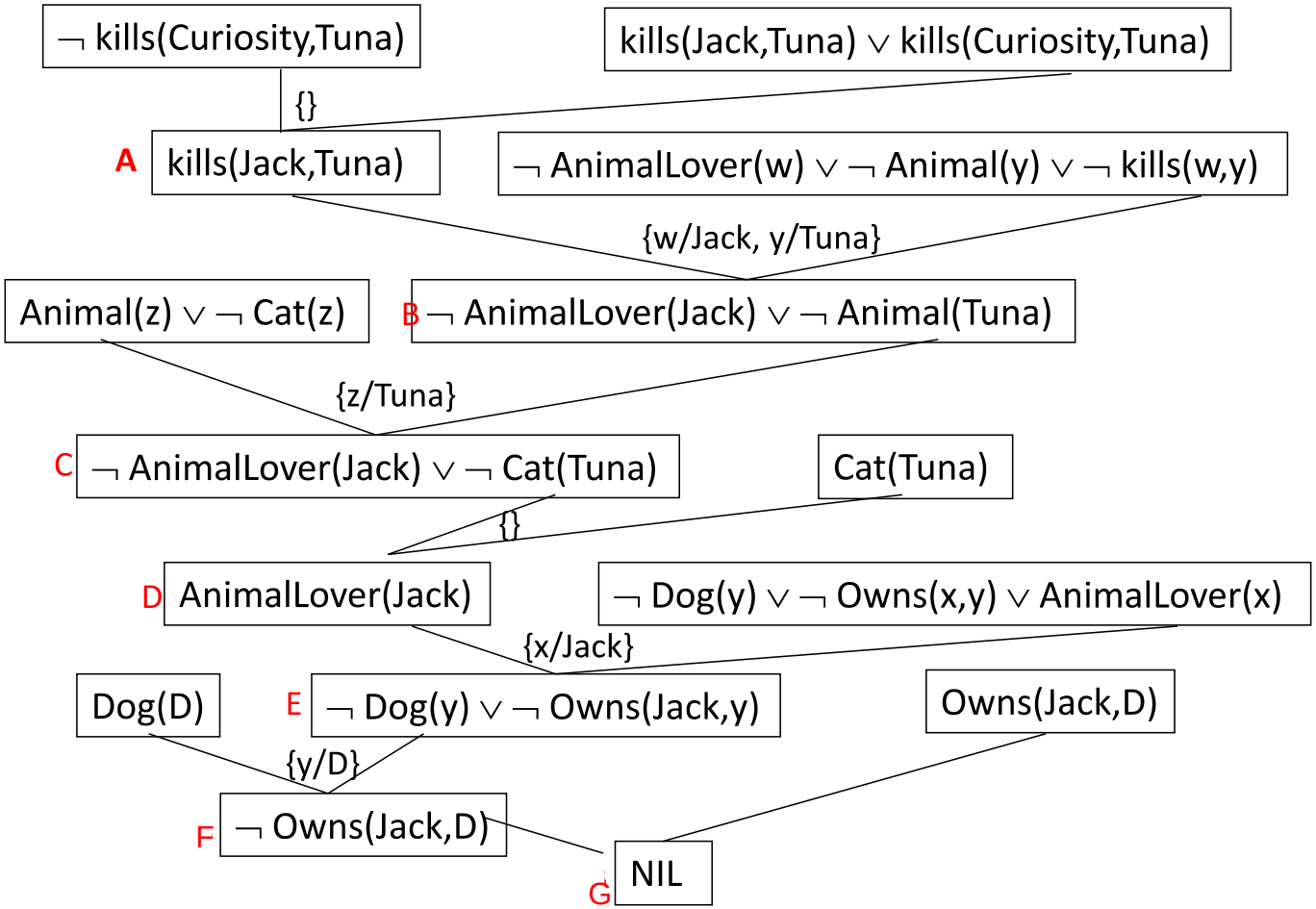
$Kills(Jack, Tuna) \vee Kills(Curiosity, Tuna)$

$Cat(Tuna)$

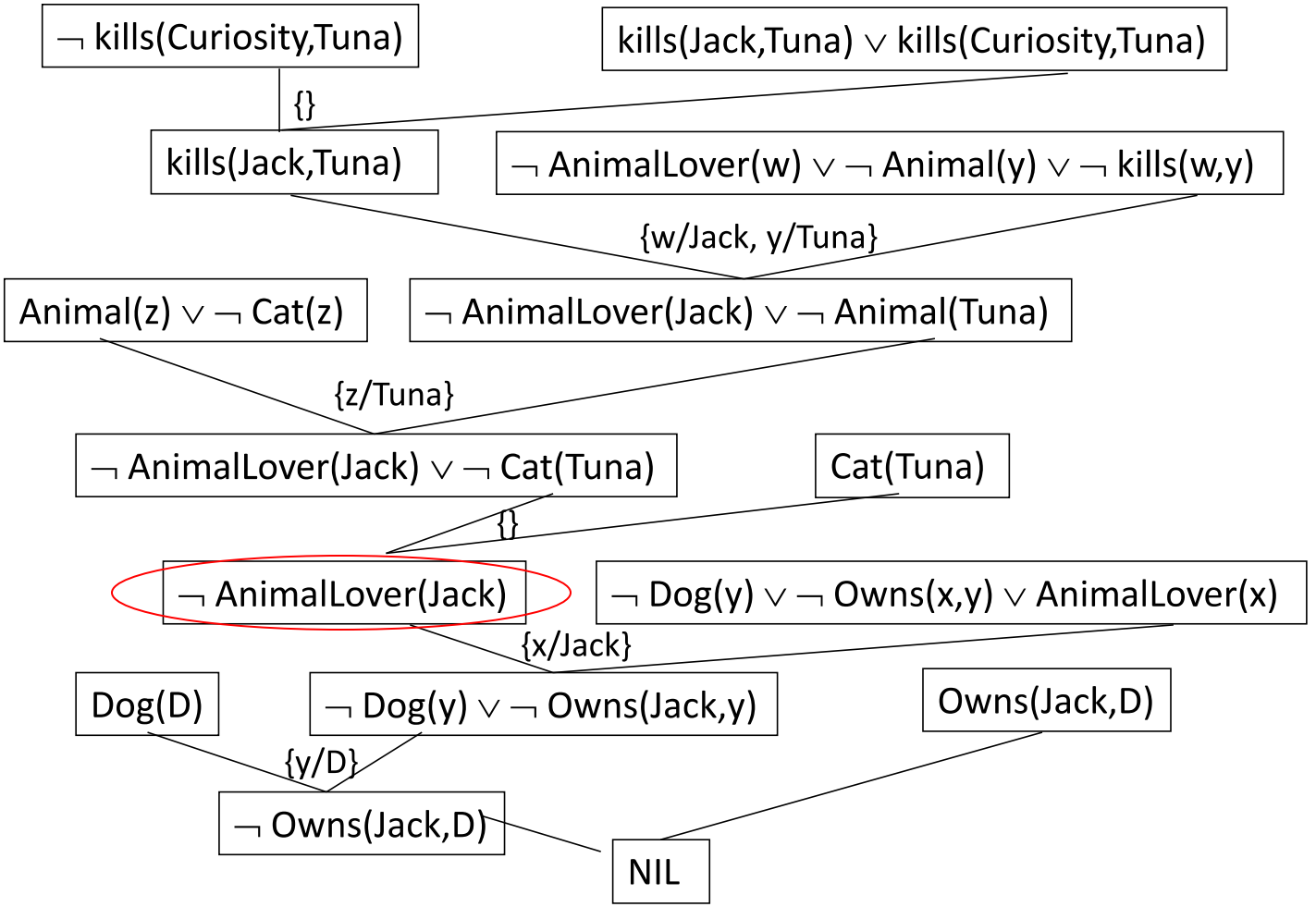
$\neg Cat(z) \vee Animal(z)$

$\neg Kills(Curiosity, Tuna)$

Example: Find the mistake!



Example: Correct proof



Resolution Strategies

Heuristics that impose a sensible order on the resolutions we attempt

- **Unit resolution**: prefer to perform resolution if one clause is just a literal – yields shorter sentences.
- **Set of support**: identify a subset of the KB (hopefully small); every resolution will take a clause from the set and resolve it with another sentence, then add the result to the set of support.
  - Can make inference incomplete!
- **Input resolution**: always combine a sentence from the query or KB with another sentences. Not complete in general.

Simple problem

- Schubert Steamroller:
  - Wolves, foxes, birds, caterpillars, and snails are animals and there are some of each of them. Also there are some grains, and grains are plants. Every animal either likes to eat all plants or all animals much smaller than itself that like to eat some plants. Caterpillars and snails are much smaller than birds, which are much smaller than foxes, which are much smaller than wolves. Wolves do not like to eat foxes or grains, while birds like to eat caterpillars but not snails. Caterpillars and snails like to eat some plants.
- Prove: there is an animal that likes to eat a grain-eating animal
- Some of the necessary logical forms:
  - $\forall$ x (Wolf(x) $\rightarrow$ animal(x))
  - $\forall$ x $\forall$ y ((Caterpillar(x) $\vee$ Bird(y)) $\rightarrow$ Smaller(x,y)
  - $\exists$ x bird(x)
- Requires almost 150 resolution steps (minimal)

Proofs can be lengthy!

Properties of knowledge-based systems

Advantages

1. Expressibility*: Human readable
2. Simplicity of inference procedures*: Rules/knowledge in same form
3. Modifiability*: Easy to change knowledge
4. Explainability: Answer "how" and "h y" questions.
5. Machine readability
6. Parallelism*

Disadvantages

1. Difficulties in expressibility
2. Undesirable interactions among rules
3. Non-transparent behavior
4. Difficult debugging
5. Slow
6. Where does the knowledge base come from???

Applications of FOL

- Expert systems
- Prolog: a logic programming language
- Production systems
- Semantic nets
- Automated theory proving
- **Planning**

What is planning?

- A **plan** is a collection of actions for performing some task.

  e.g., assembling a new IKEA desk

- Use **a subset of FOL** to describe the problem
  - What actions are we allowed to take? (put two pieces together, tighten a screw, …)
  - When are we allowed to take them? (box must be opened, piece A and piece B must already be assembled, …)
  - What is the initial state? (all the pieces are in the box)
  - What is the goal state? (furniture is assembled)

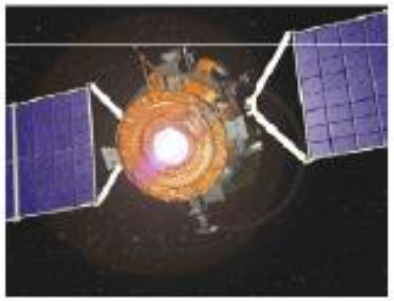
Applications

Games

Military Logistics

Manufacturing

Robots

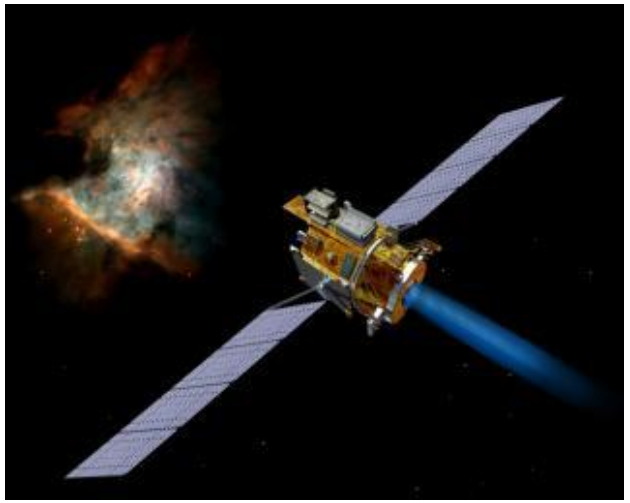


Autonomous Spacecraft

Pictures from: *http://www.stanford.edu/class/cs227/Lectures/lec16.pdf*

NASA's Deep Space 1 (1998)

- Launched in Oct. 1998 to test technologies and perform flybys of asteroid Braille and Comet Borrelly.
- Autonav system used for autonomous navigation and finding imaging targets.
- Remote Agent system used to perform automatic fault detection and self-repair.
- First spacecraft to be controlled by AI system without human intervention!



www.jpl.nasa.gov

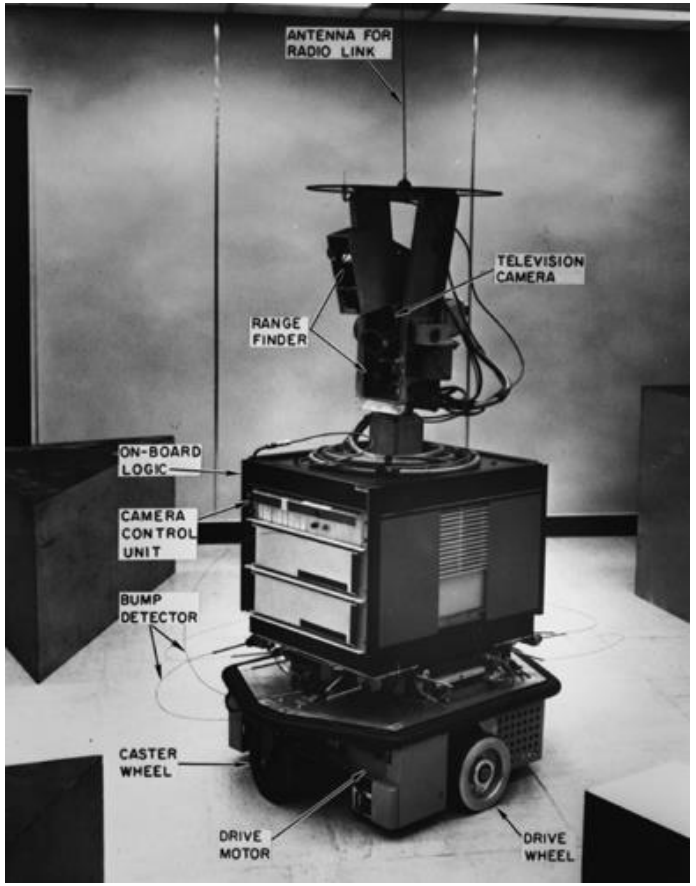
Planning as search

- Planning problem is described like a search problem (states, operators, goals), but **the problem representation is a lot more structured**

Search	Planning
States are **atomic**	States and goals are **logical sentences**
Actions are **atomic**	Actions are described in logic by **preconditions** and **outcomes**

Shakey: "The first electronic person"

- Designed in 1966, at the Stanford Research Institute
- Able to plan and execute simple tasks related to navigation and object manipulation
- **Needed a language** in which to express planning tasks
  - The **STRIPS language**



http://www.ai.sri.com/shakey/

Let's Help Shakey Plan to Move

- How can Shakey move from point A to point B?
  - It must be in point A.
  - Nothing must be impeding Shakey.
  - Point B must be clear.
- What happens after Shakey moves from point A to point B?
  - Point A is now clear.
  - Point B is no longer clear.
  - Shakey is now in point B.
- Let's do all this more formally!

STRIPS (Stanford Research Institute Planning System)

- **Domain**: set of typed objects represented as propositions.

- **States**: represented as **first-order predicates over objects**.

  Closed-world assumption:

  - everything not stated is **false**
  - only objects in the world are the ones defined.

- **Operators**: defined in terms of:
  - **Preconditions**: when can the action be applied?
  - **Effects**: what happens after the action?

  (No explicit description of how the action should be executed.)

STRIPS* representations

- **States** are represented as conjunctions of predicates:

  $In(robot, room) \wedge Closed(door) \wedge \ldots$

- **Goals** are represented as conjunctions of predicates:

  $In(robot, r) \wedge In(Charger, r)$

- **Operators**:
  - Name: $Go(x, y)$
  - Preconditions are represented as conjunctions:

    $At(robot, x) \wedge Path(x, y)$

  - Postconditions are represented as conjunctions:

    $At(robot, y) \wedge \neg At(robot, x)$

- Variables (e.g. *x, y, r*) are **typed**; they can only be instantiated with objects of correct type.

*R&N uses a very similar representation called PDDL

STRIPS operator representation

- An **action schema** defines the following for an operator:
- {*Name*, *Preconditions*, *Effects*}

- Preconditions are conjunctions of positive literals

- Effects/Postconditions are represented in terms of:
  - **Add-list**: list of propositions that become true after the action.
  - **Delete-list**: list of propositions that become false after the action.
  - e.g. $At(robot, y) \wedge \neg At(robot, x)$

Add the positive literals

Delete the negative literals

Semantics of an Action

- If the precondition is false in a world state:
  - the action does not change anything (since it cannot be applied).
- If the precondition is true:
  - Delete the items on the Delete-list.
  - Add the items on the Add-list.

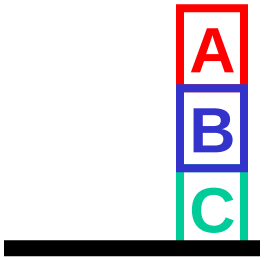
  Order of operations is important here!
- This is a very restricted language, which means we can do efficient inference.

Example: Move action

- **Action**:
  - *Move(object, from, to)*
- **Preconditions**:
  - *At(object, from), Clear(to), Clear(object)*
    - There are implicit conjunctions here.
- **Effects**:
  - Delete-list: *At(object, from)*, *Clear(to)*
  - Add-list: *At(object, to)*, *Clear(from)*
    - In logical form: $At(object, to) \wedge Clear(from) \wedge \neg At(object, from) \wedge \neg Clear(to)$

Welcome to the Blocks World!

Initial state



Goal state

Initial state = On(A,table) ∧ On(B,table) ∧ On(C,table) ∧ Clear(A) ∧ Clear(B) ∧ Clear(C)

Goal state = On(A,B) ∧ On (B,C)

Exercise: Fill in the preconditions and effects of these actions

Action = Move(b,x,y) Move b from x to y, where y is not the table.

Precondition =

Effect =

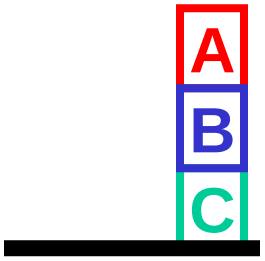
Action = MoveToTable(b,x) Move b from x to the table.

Preconditions =

Effect =

Welcome to the Blocks World!

Initial state



Goal state

Initial state = On(A,table) ∧ On(B,table) ∧ On(C,table) ∧ Clear(A) ∧ Clear(B) ∧ Clear(C)

Goal state = On(A,B) ∧ On (B,C)

Action = Move(b,x,y)

Precondition = On(b,x) ∧ Clear(b) ∧ Clear(y)

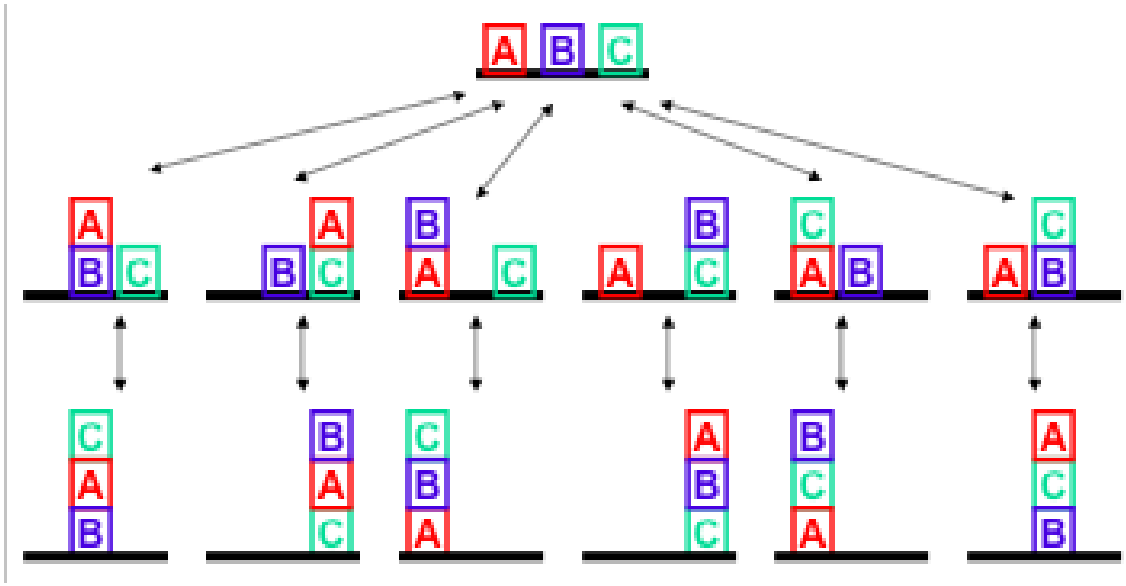
Effect = On(b,y) ∧ Clear(x) ∧ ¬On(b,x) ∧ ¬Clear(y)

Action = MoveToTable(b,x)

Preconditions = On(b,x) ∧ Clear(b)

Effect = On(b,table) ∧ Clear(x) ∧ ¬On(b,x)

STRIPS state transitions



Exercise

Write out a plan that actually achieves the goal state

Pros and cons of STRIPS?

- **Pros**:
  - Since it is restricted, inference can be done efficiently.
  - All operators can be viewed as simple *deletions* and *additions* of propositions to the knowledge base.

- **Cons**:
  - Assumes that a small number of propositions will change for each action (otherwise operators are hard to write down, and reasoning becomes expensive.)
  - Limited language (preconditions and effects are expressed as conjunctions), so not applicable to all domains of interest.

Two basic approaches to planning

1. **State-space planning** works at the level of states and operators.
   - Finding a plan is formulated as a search through state space for a path from the start state to the goal state(s).
   - Most similar to constructive search.

2. **Plan-space planning** works at the level of plans.
   - Partial order planning – read R&N 10.4.4

Progression (forward) planning

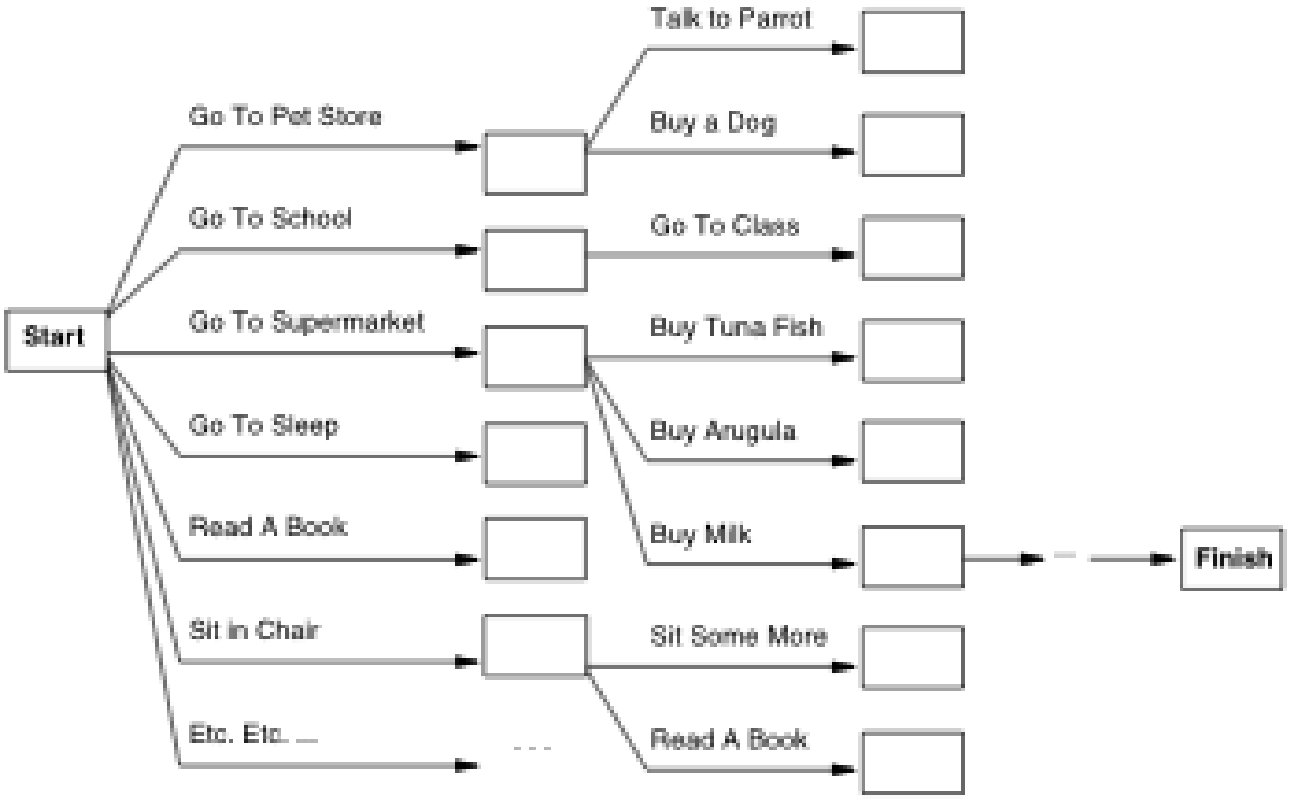
Starting from the initial state, do:

1. Determine all operators that are applicable in the start state by examining preconditions.
2. Ground the operators, by replacing any variables with constants.
3. Choose an operator to apply.
4. Determine the new content of the knowledge base, based on the operator description (apply the effects from the add- and delete lists).

Repeat until goal state is reached.

Example: Supermarket domain

- We're at home, but we need to buy milk, bananas, and a cordless drill (don't ask).



Example: Supermarket domain

- In the start state we have *At(Home)*, which allows us to apply operators of the type *Go(x,y)*.
- The operator can be instantiated as *Go(Home, HardwareStore)*, *Go(Home, GroceryStore)*, *Go(Home, School)*, …
- If we choose to apply *Go(Home, HardwareStore)*, we will delete from the KB *At(Home)* and add *At(HardwareStore)*.
- The new proposition enables new actions, e.g. *Buy*

- Note that now there are a lot of possible operators to consider!

Regression planning

- Pick actions that satisfy (some of) the goal propositions.
- Make a new goal, containing the preconditions of these actions, as well as any unsolved goal propositions.
- Repeat until the goal set is satisfied by the start state.

Example: Supermarket domain

- In the goal state we have *At(Home)* $\wedge$ *Have(Milk)* $\wedge$ *Have(Bananas)* $\wedge$ *Have(Drill)*
- The action *Buy(Milk)* would allow us to achieve *Have(Milk)*.
- To apply this action we need to have the precondition *At(GroceryStore)*, so we add it to the set of propositions we want to achieve.
- Similarly, we want to achieve *At(HardwareStore)*.

Note that in this case, the order in which we try to achieve these propositions matters!

State-space planning

- **Progressive planners** reason from the start state, trying to find the operators that can be applied. (-> match preconditions)
  - Analogy: forward search!

- **Regression planners** reason from the goal state, trying to find the actions that will lead to the goal. (-> match effects)
  - Analogy: Backward search!

In both cases, the planners work with **sets of states**, instead of using individual states as in straightforward search.

Analysis of STRIPS planning

- STRIPS planning is **SOUND**.
  - Only legal plans will be found.
- STRIPS planning is **NOT COMPLETE**.
  - Once a subgoal ordering is selected, no backtracking is allowed.
- STRIPS planning is **NOT OPTIMAL**.
  - No guarantee of finding shortest possible plan.
- STRIPS planning is **EXPENSIVE**. Typically NP-hard or worse.

Some state-of-the-art classical planners

- SATPlan: Planning as satisfiability (Kautz & Selman, 1996)
- Heuristic-search planning (Bonet & Geffner, 1998)
- GraphPlan (Blum & Furst, 1995)
- Fast Forward planning (Hoffman & Nebel, 2001)
  - Use hill-climbing to get near a good solution, then best-first-search.
- Hierarchical task network planning
  - Decompose complex planning problem into a hierarchy of smaller planning tasks.
- Many more!

Planning as logic

- Planning is very much like searching over sets of states, but the problem representation is more structured.

	Search	*Planning*
States	Data structures	Logical sentences
Actions	Code	Preconditions/outcomes
Goal	Goal test	Logical sentence (conjunction)
Plan	Sequence from S_0	Constraints on actions

Key idea: describe states and actions in propositional logic and use forward/backward chaining to find a plan.

Planning as satisfiability: SatPlan

- Introduced by Kautz and Selman, 1990s, very successful method over the years.
- Take a description of a planning problem and generate all possible literals at all time slices.
- Generate a humongous SAT problem.
- Use a state-of-the-art SAT solver (e.g WalkSAT) to get a plan.
- Randomized SAT solvers can be used as well.

Analysis of SatPlan

- Optimality: Yes! (Assuming exact SAT solution.)
- Completeness: Yes!
- Complexity:
  - Clearly NP-hard (as it can be seen as SAT in finite-length plan case).
  - But actually worse (PSPACE) if we let plan duration vary.

Heuristic-search planning

- Don’t want domain-specific heuristics.
- Define heuristics based on planning problem itself.
  - Can we derive an **admissible** heuristic for search in planning domains?
  - Technique we've seen before: solve **relaxed** problem!

- Solving the problem becomes much easier in the relaxed case; number of steps to goal in this heuristic can be used as part of A* search
  - Historical note: A* search came from the same research project responsible for STRIPS and Shakey!

Two Heuristics

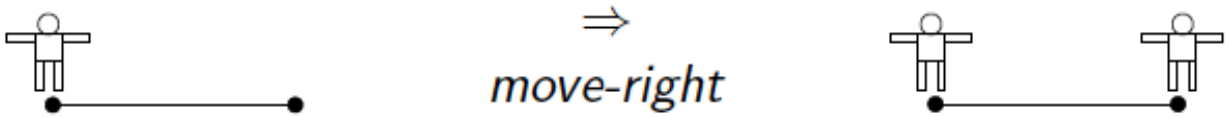
1. Ignore Preconditions

- Assume that we can also apply an operator

e.g., you can always move to point B, no matter where you are

2. Ignore Delete lists (in Effects)

- Assumes that when something is achieved, it stays achieved.



- Solve this relaxed version and use this as the heuristic.

Image from: *http://icaps09.uom.gr/tutorials/tut1.pdf*

Problems in the real world

- **Incomplete information**
  - Unknown predications, e.g., *Intact(Spare)*
  - Disjunctive effects, e.g. *Inflate(x)* causes *Inflated(x)* according to the KB, but in reality it actually causes *Inflated(x) v SlowHiss(x) v Burst(x) v BrokenPump v …*
- **Incorrect information**
  - Current state it incorrect, e.g., spare NOT intact.
  - Unanticipated outcomes (missing / incorrect postconditions) of operators, that lead to failure.
- **Qualification problem**
  - Can never finish listing all the required preconditions and possible conditional outcomes of actions.

The real world: Some solutions

- **Conditional (contingency) planning:**
  - Plans include observation actions which obtain information.
  - Sub-plans are created for each contingency (eash possible outcome of the observation actions).

  E.g. Check the tire. If it is intact, then we're ok, otherwise there are several possible solutions: Inflate, Call-CAA, …
  - Expensive because it plans for many unlikely cases.

- **Monitoring / Replanning:**
  - Assume normal states, outcomes.
  - Check progress during execution, replay if necessary.

  In general, some monitoring / replanning is highly useful.