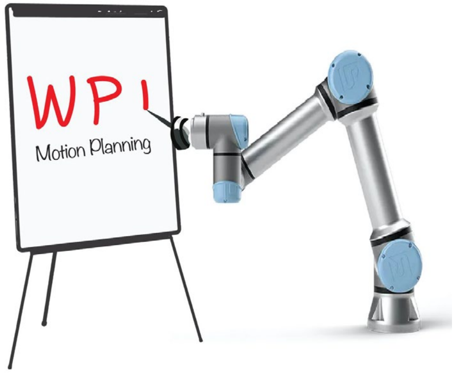
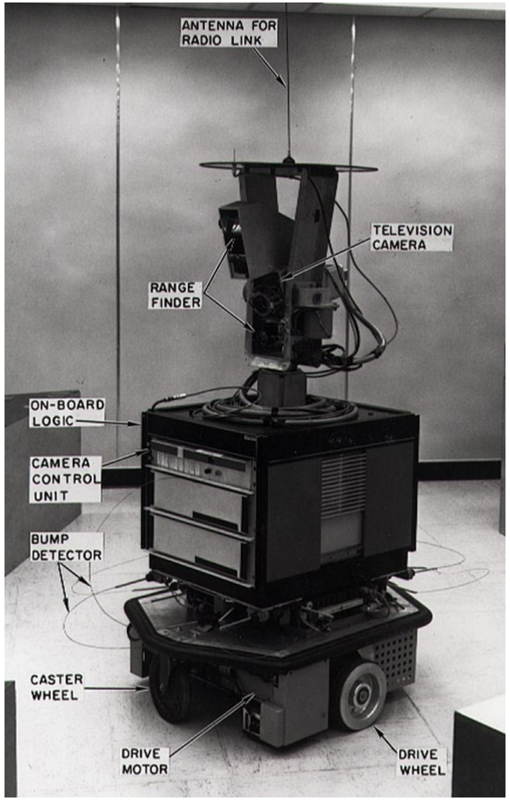
RBE550
Motion Planning
Task Planning



Constantinos Chamzas
www.cchamzas.com
www.elpislab.org

Motivation

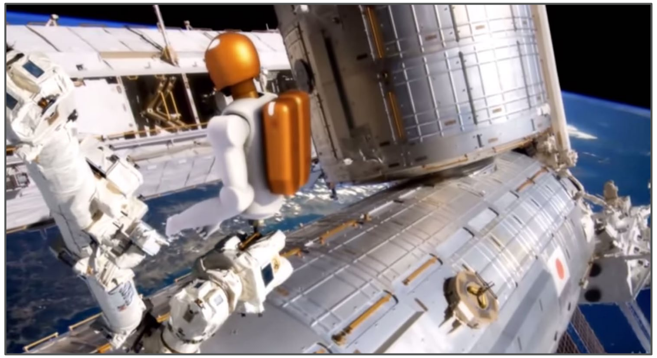
- This is all because of Shakey the Robot
- Check out:
- https://youtu.be/7bsEN8mwUB8
- https://youtu.be/GmU7SimFkpU



Shakey the Robot
SRI: 1966 - 1972

Motivation

- Want robot to perform complex tasks autonomously
- e.g., Household robot to clean the house
- e.g., Robonaut to fix and replace components on the exterior of ISS
- Given high-level task description, have the robot achieve this task



Motivation

So far in this course:

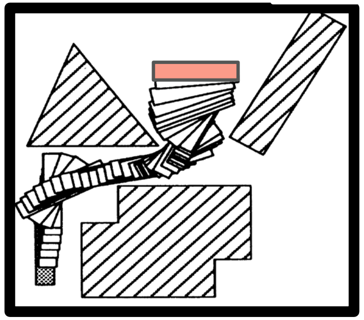
- ➢A-to-B motion planning
- ➢Geometric / kinematic – Continuous!
- ➢Motion planning problem

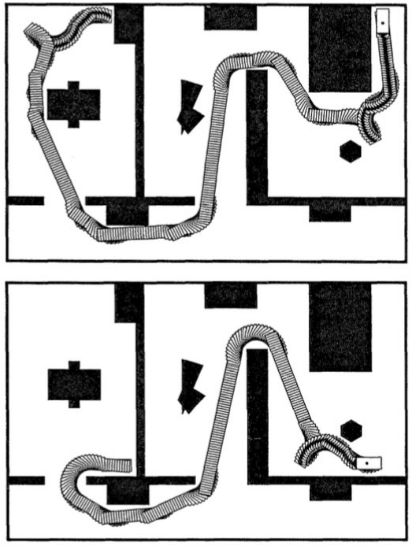
Given:

- ➢Start configuration
- ➢Goal set
- ➢Distance metric

Find:

- ➢A continuous path to goal



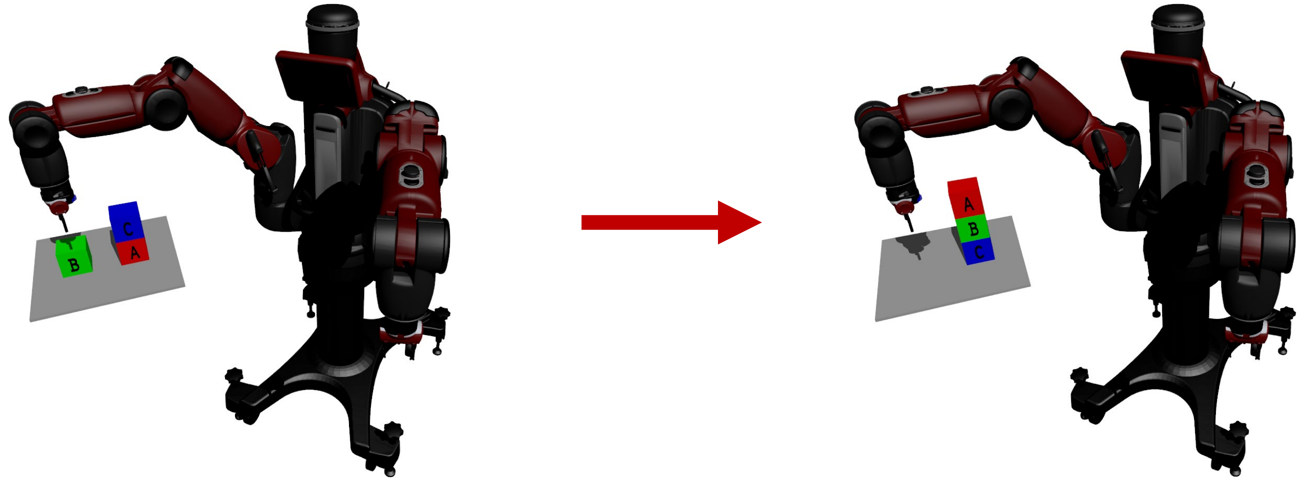


Motion Planning

- So far in this course …
  - can plan for this robot to move its end-effector from A to B

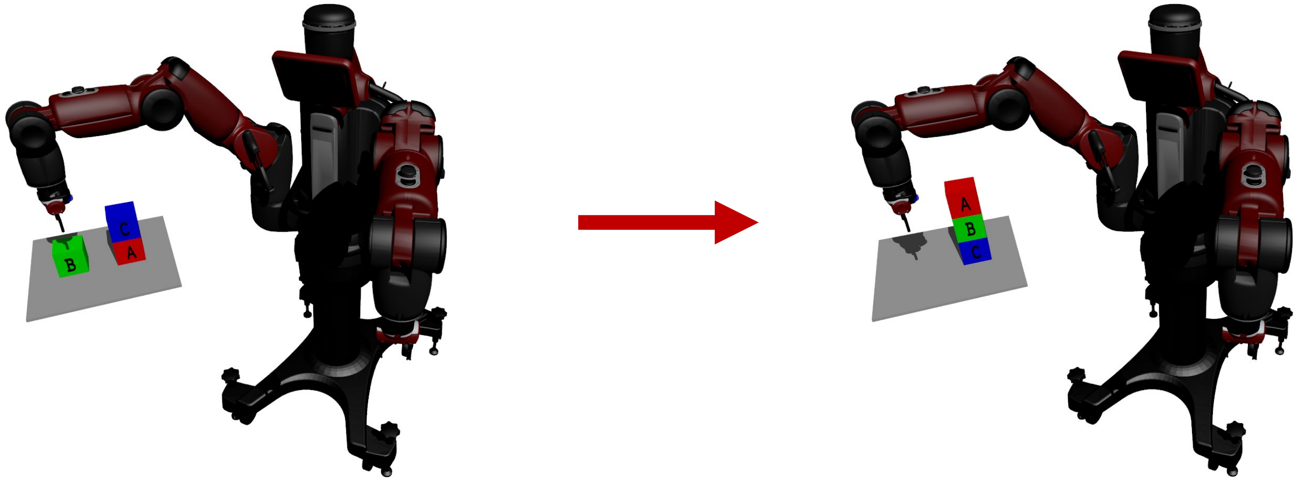
Task and Motion Planning

- Now, we want the robot to perform a complex task
  - e.g., stack up the objects such that A is on top of B and B is on top of C.
- In short, tell robot where everything is and where everything needs to end up. Let robot figure it out.



Task and Motion Planning

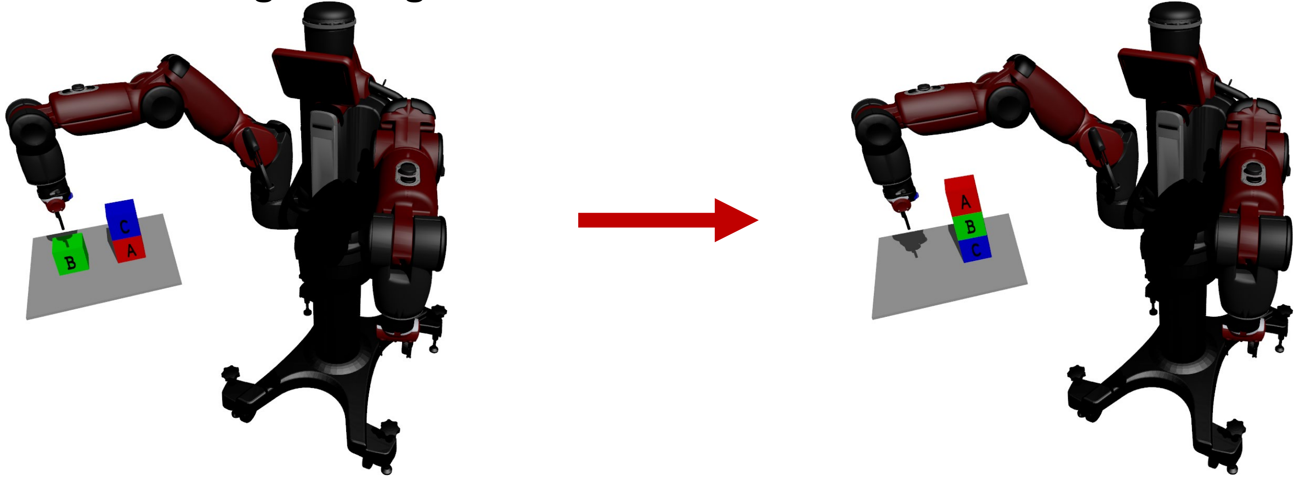
Informal		General Case
Tell robot where everything is	❯	Initial state



Task and Motion Planning

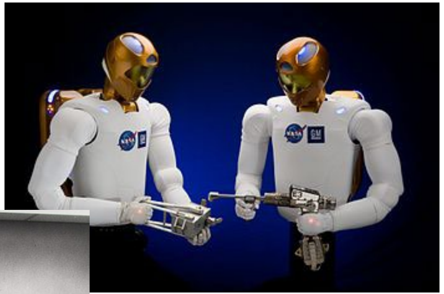
Informal		General Case
Tell robot where everything is	❯	Initial state
Tell it where everything needs to end up	❯	Goal state(s)
Let the robot figure it out	❯	Transitions between states

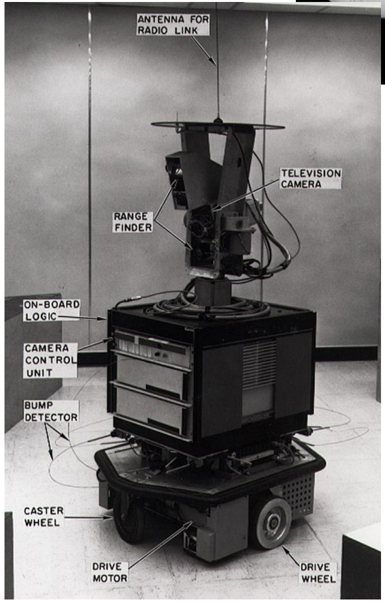
Design an algorithm to find a solution if one exists.



Task and Motion Planning

- How to approach this problem?
  - Define a representation of the problem that can be automatically solved by a robot.
  - Need some formal way to describe the state of the world and how the robot can interact with the world.
- In this lecture, focus on Task Planning:
  - assume the motion planning problem is trivial (no fun!)





Overview

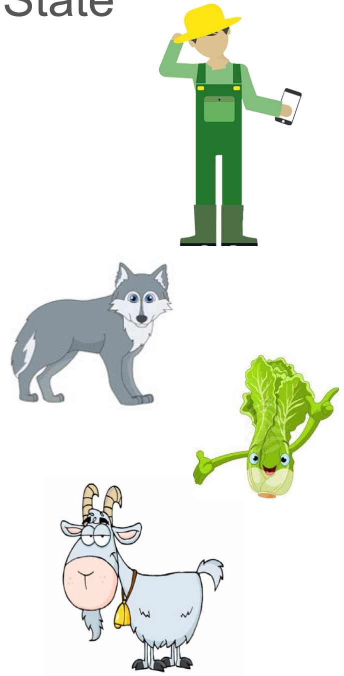
- Example of task planning with discrete state-space search
- Overview of formal task representation
- Heuristic search methods for task planning

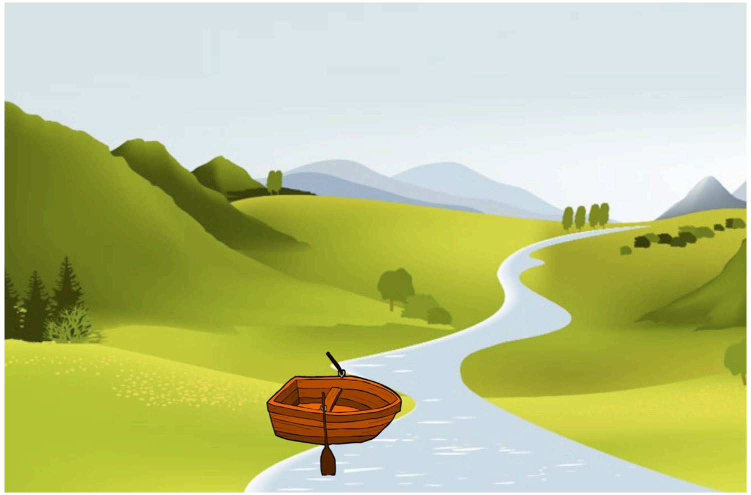
Planning Puzzle

- The farmer has a goat, a cabbage and a wolf with him and needs to cross a stream. He has a boat, but it can only hold him and one of the objects. If he leaves the goat alone with the cabbage the cabbage will be eaten and if he leaves the wolf alone with the goat the goat will be eaten. How can he get across the stream with all objects?

Planning Puzzle

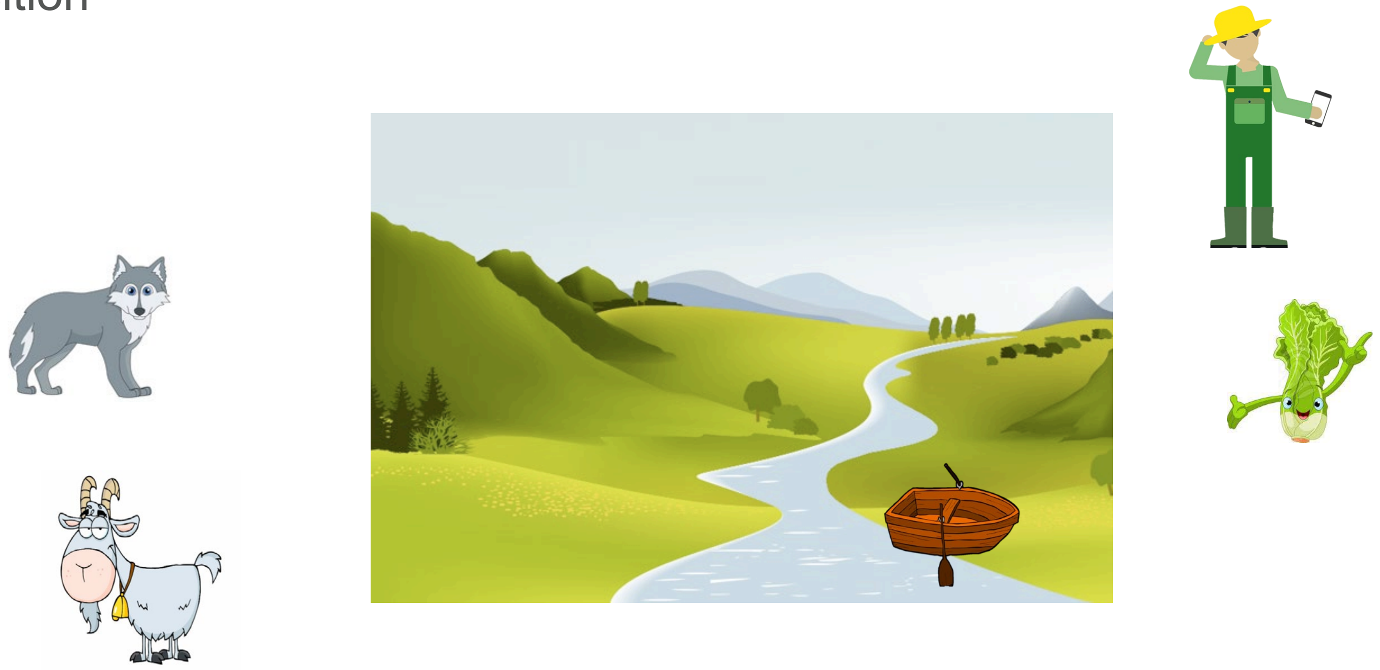
- Initial State





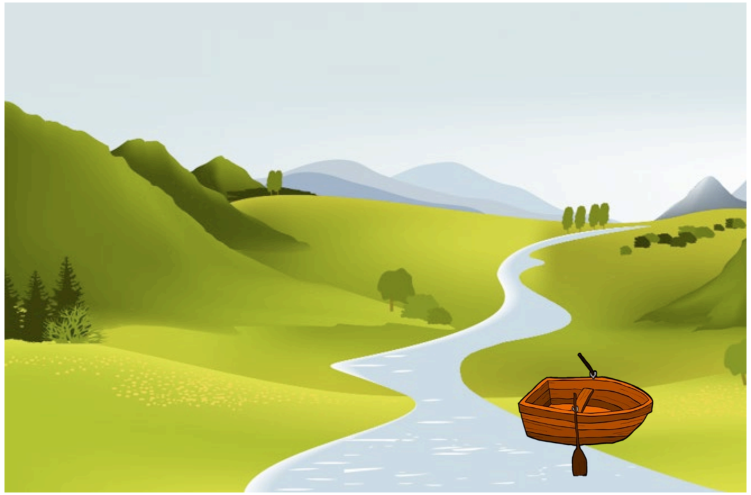
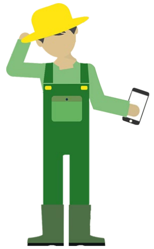
Planning Puzzle

- Transition



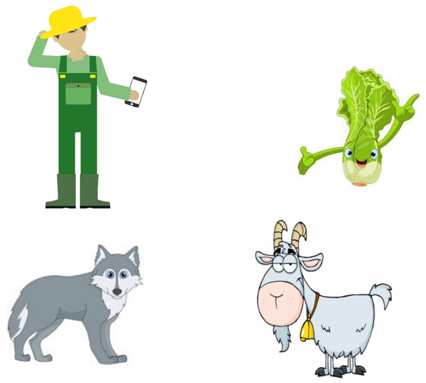
Planning Puzzle

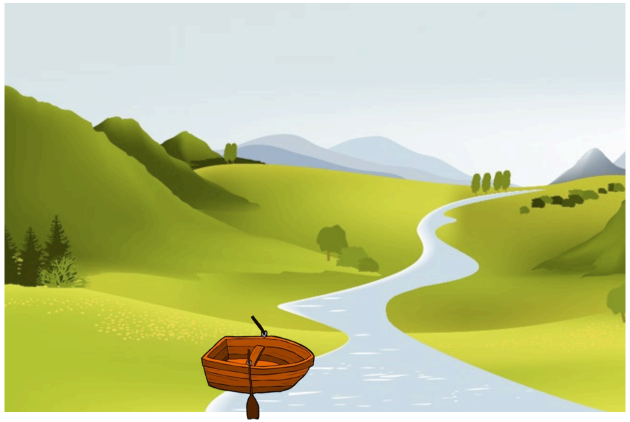
- Transition

Planning Puzzle

- How to represent this problem?
- Many representations are possible
- What makes a good representation?

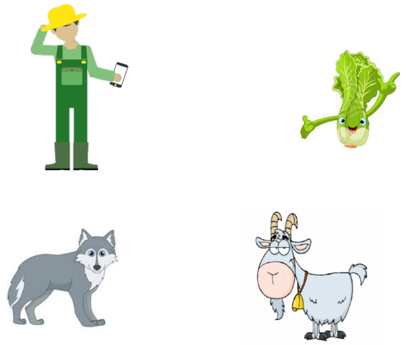


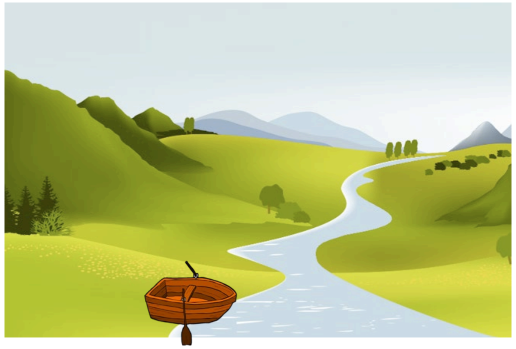


Planning Puzzle: representation

- How to represent this problem?
- Many representations are possible
- What makes a good representation?

$$\begin{tikzcd} & \text{(wolf, goat, cabbage)} \arrow[dl] \arrow[dr] & \\ (0, 0, 0) & & (1, 1, 1) \end{tikzcd}$$





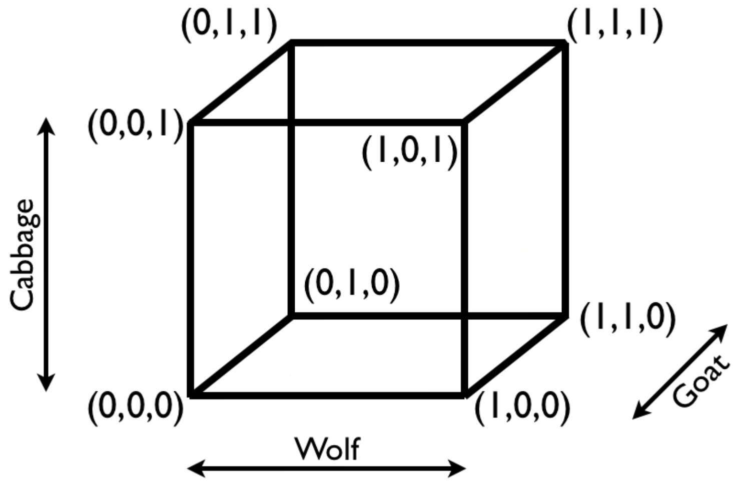
Planning Puzzle: representation possibilities

- How to represent this problem?
- Many representations are possible
- What makes a good representation?

Vector	State
(0,0,0)	Our initial state, with all three on the starting side of the river.
(1,0,0)	The wolf has crossed the river, but not the goat or cabbage.
(0,1,0)	The goat has crossed the river, but not the wolf or cabbage.
(0,0,1)	The cabbage has crossed the river, but not the wolf or goat.
(1,1,0)	The wolf and the goat have crossed the river, but not the cabbage.
(0,1,1)	The goat and the cabbage have crossed the river, but not the wolf.
(1,0,1)	The wolf and cabbage have crossed the river, but not the goat.
(1,1,1)	Our desired state, where all three have crossed the river.

Planning Puzzle

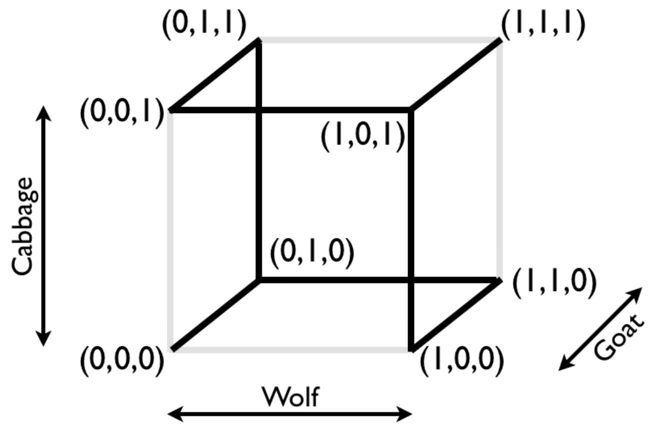
- State space representation



Planning Puzzle

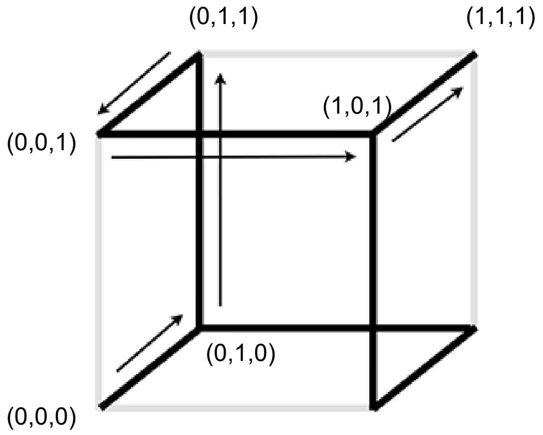
- State space representation

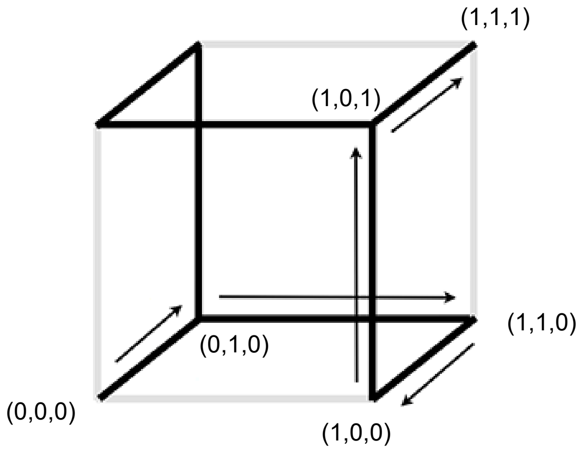
Path Removed	Rationale
(0,0,0) to (1,0,0)	This moves the wolf across, leaving the goat with the cabbage.
(0,0,0) to (0,0,1)	This moves the cabbage across, leaving the wolf with the goat.
(0,1,1) to (1,1,1)	This moves the wolf across, leaving the goat with the cabbage.
(1,1,0) to (1,1,1)	This moves the cabbage across, leaving the wolf with the goat.



Planning Puzzle

- State space search





1. Move goat to other side.
2. Move cabbage to other side.
3. Move goat back.
4. Move wolf to other side.
5. Move goat to other side.

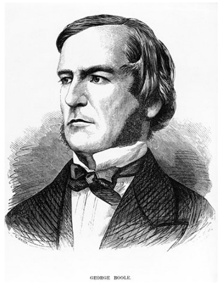
1. Move goat to other side.
2. Move wolf to other side.
3. Move goat back.
4. Move cabbage to other side.
5. Move goat to other side.

Overview

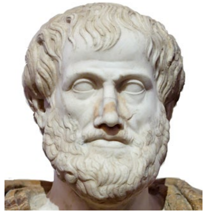
- Example of task planning with discrete state-space search
- Overview of formal task representation
    - Boolean Logic
    - First-order Logic
    - Planning Domain Definition Language (PDDL)
    - Temporal Logics
- Heuristic search methods for task planning

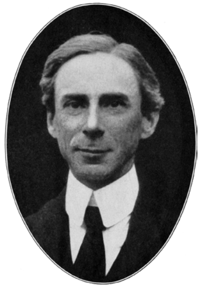
Logic

- Logic is one of the oldest intellectual disciplines in human history
- It dates back to Aristotle. It has been studied through the centuries by people like Leibniz, Boole, Russell, Turing, and many others.
- It is still a subject of active investigation today



Boole


Aristotle


Russell

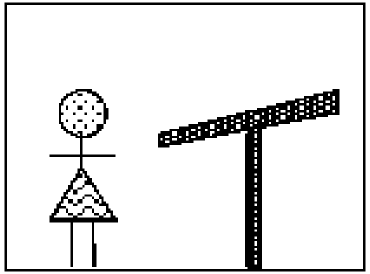
Leibniz

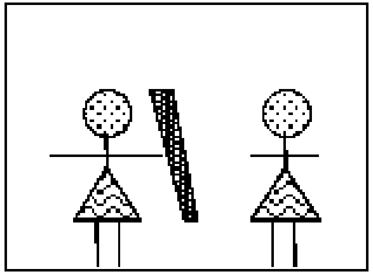
Turing

Symbolic Logic (Formalization)

English (natural) language is problematic. Sentences can

- be complex
- be ambiguous
- lead to errors in reasoning
  - Example "There's a girl in the room with a telescope"
  - Room containing a telescope? OR
  - Girl holding a telescope?





Symbolic Logic (Formalization)

- Illustration of errors that arise in reasoning with sentences in natural language

- The following works (because of transitivity)

Champagne is better than beer.

Beer is better than soda.

Therefore, champagne is better than soda.

- Does this work?

Soda is better than nothing.

Nothing is better than champagne.

Therefore, soda is better than champagne.

- Symbolic Logic eliminates these difficulties through the use of a formal language for encoding information

- Given the syntax and semantics of this formal language, we can give a precise definition for the notion of logical conclusion

Symbolic Logic

Symbolic logic encodes logical reasoning by means of symbols

Advantages:

- Precise and unambiguous alternative to natural language
- Enables separating out pure logical reasoning from domain-specific reasonings
- Logical reasoning can be carried out mechanically by “symbol-pushing” or “symbol processing”
    - Can lead to AUTOMATED REASONING – an important branch of computer science

Symbolic Logic

- Symbolic logic provides a language for representing statements and arguments and expressing world knowledge
- This language has 2 aspects: syntax and semantics (interpretation).

- Subclasses of symbolic logic:
    - Propositional logic (Boolean logic)
    - First order logic
    - High order logics
    - Modal logics (temporal logics)

Symbolic Logic

- Applications of symbolic logic
    - Precise specification (hardware, software, electromechanical systems)
    - Rigorous verification of systems (complementary to testing & simulation)
    - Rigorous synthesis of systems (automated control synthesis)
    - Programming language design (Prolog, Datalog)
    - Theoretical computer science (complexity theory)

Propositional Logic

- Propositional Logic is concerned with propositions and their interrelationships
- Roughly speaking, a proposition is a possible condition of the world that is either **true** or **false**
  - e.g., “*this course is awesome*”
- A declarative statement is a collection of basic propositions connected together by some logical connectives
  - e.g., “*this course is awesome, and I love it*”
- Statements expressed in propositional logic are called **formulas**

Propositional Logic

- **Propositional symbols:** lower case letters - $p, q, r, \ldots$

- **Truth constants:** two special propositional constant symbols
  - *true* $(T, \top, 1)$
  - *false* $(F, \bot, 0)$

- **Logical connectives** (Boolean operators):

Negation: ¬ (not)	$\neg\, a$
Conjunction: ∧ (and)	$a \wedge b$
Disjunction: ∨ (or)	$a \vee b$
Implication: → (if… then…)	$a \rightarrow b$
Equivalence: ↔ (… if and only if …)	$a \leftrightarrow b$

Syntactically Valid Sentences

- Syntax of PL can be defined recursively as

$$\phi \coloneqq p \mid \neg\phi \mid \phi \wedge \phi \ \ldots$$

1. All proposition symbols and constants T and F are syntactically valid formals

2. If ϕ_1 and ϕ_2 are syntactically valid formulas, then

   $$\neg\phi_1, \qquad \phi_1 \wedge \phi_2, \qquad \phi_1 \vee \phi_2, \qquad \phi_1 \rightarrow \phi_2, \qquad \phi_1 \leftrightarrow \phi_2$$

   are all syntactically valid formulas

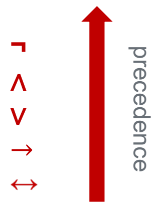
3. Only those formulas obtained using the above clauses are syntactically valid formulas
   - Examples:
     - Syntactically valid formulas
       - $\big(p \wedge (q \vee r)\big)$
       - $\Big(\big(p \vee (q \wedge T)\big) \rightarrow \big((\neg r \wedge q) \leftrightarrow p\big)\Big)$
     - Syntactically invalid formulas
       - $\vee\, q$
       - $(p \wedge \vee\, q)$
       - $(p \neg q)$
       - $(pq \vee)$

Syntactically Valid Sentences

- What to do with too many parentheses?

$$\Big((p \lor ((q \land r) \land T)) \to ((\neg r \land (q \lor p)) \leftrightarrow p)\Big)$$

  - Drop out-most parentheses
  - Use (standard) operator precedence
    - $\neg$ operator has higher precedence than $\land$
    - $\land$ has higher precedence than $\lor$
    - $\lor$ has higher precedence than $\to$ and $\leftrightarrow$.



- Examples:

- $\neg p \lor q$ means $((\neg p) \lor q\ )$
- $p \lor q \land r$ means $(p \lor (q \land r)\ )$

- $p \lor q \land r \land T \to (\neg r \land (q \lor p) \leftrightarrow p)$

means

$$\Big((p \lor ((q \land r) \land T)) \to (((\neg r) \land (q \lor p)) \leftrightarrow p)\Big)$$

Semantics

- The semantics involves giving interpretations for syntactic formulas
- Interpreting formulas as statements is concrete
- Interest is only in the truth values of formulas, i.e., whether a formula is ***true*** or ***false***
- Abstract semantics that assigns truth values to formulas
- Truth values are ***true***, ***false*** – two valued semantics
- There is a systematic way of obtaining truth values of formulas based on the syntax
    - Assign to $T(1)$, $F(0)$ the values true - false respectively
    - Assign arbitrary truth values to every propositional symbols
    - Interpret the logical connective as operators over truth values

Truth Table

- The operator for connectives are given by the **truth table**

p	q	$\neg p$	$p \wedge q$	$p \vee q$	$p \rightarrow q$	$p \leftrightarrow q$
1	1	0	1	1	1	1
1	0	0	0	1	0	0
0	1	1	0	1	1	0
0	0	1	0	0	1	1

- Given this interpretation, a formula is assigned a truth value as follows
  - If the sub-formulas ϕ_1 and ϕ_2 are assigned truth values, then the truth values for the formulas $\neg\phi_1$, $(\phi_1 \wedge \phi_2)$, $(\phi_1 \vee \phi_2)$, $(\phi_1 \rightarrow \phi_2)$, and $(\phi_1 \leftrightarrow \phi_2)$ are evaluated using the truth tables for these operators

Propositional Logic

- Example:

$$\text{a=0, b=0, c=1, d=1}$$
$$((\text{a}\vee b) \wedge (c \wedge d))$$
$$((0\vee 0) \wedge (1 \wedge 1))$$
$$(0\wedge(1 \wedge 1))$$
$$(0\wedge 1)$$
$$0$$

Propositional Logic

- Example:

$$\text{a}=0,\ \text{b}=0,\ \text{c}=1,\ \text{d}=1$$
$$((\text{a}\vee b) \rightarrow (c \wedge d))$$
$$((0\vee 0) \rightarrow (1 \wedge 1))$$
$$(0\rightarrow (1 \wedge 1))$$
$$(0\rightarrow 1)$$
$$1$$

Propositional Logic

- Example (proof):

1. If it is raining, then the ground is wet.
2. The ground is wet.

---

3. *Therefore, it is raining. (Invalid inference)*

- R = it is raining; W = the ground is wet

1. $R \rightarrow W$
2. W

---

3. *R (Invalid Inference)*

Propositional Logic

- Example (proof):

1. If it is raining, then the ground is wet.
2. The ground is wet.

---

3. *Therefore, it is raining. (Invalid inference)*

- R = it is raining; W = the ground is wet

```
1.   R → W
2.   W
```

```
3.   R (Invalid Inference)
```

❖ Can test validity by checking if the conclusion can be false when the premises are true
❖ If the argument is valid the following formula must always be false, otherwise we have falsified the argument

$$(((\mathrm{R} \rightarrow W) \wedge W) \wedge (\neg R))$$

Propositional Logic

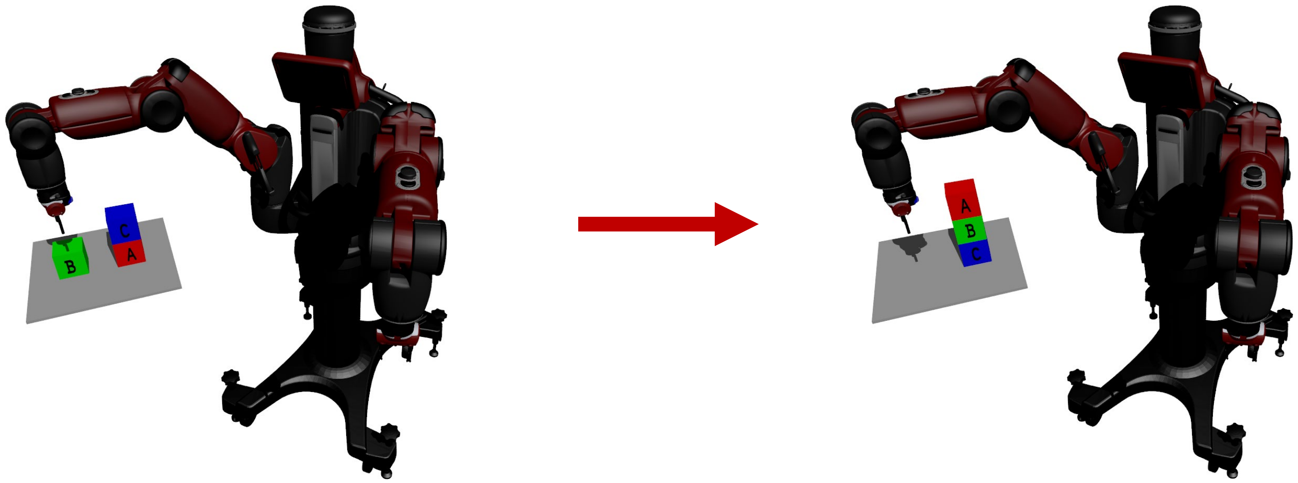
- Example (proof):

$$\mathrm{R} = 0,\ \mathrm{W} = 1$$
$$(((\mathrm{R} \rightarrow W) \wedge W) \wedge (\neg R))$$
$$(((0 \rightarrow 1) \wedge 1) \wedge (\neg 0))$$
$$((1 \wedge 1) \wedge (\neg 0))$$
$$(1 \wedge (\neg 0))$$
$$(1 \wedge 1)$$
$$1$$

  - Because the conclusion can be false while the premises are true, the argument is invalid
  - Note that we can enumerate all assignments to R and W, so we can test this mechanically

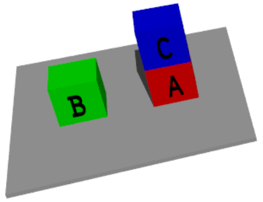
Task Specification

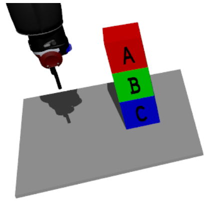
- Example:



Task Specification

- Example:





B-is-on-Table AND
C-is-on-A AND
A-is-on-Table

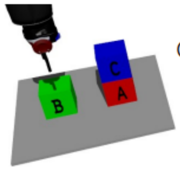
(NOT A-is-on-B) AND
(NOT B-is-on-C) AND
(NOT C-is-on-
TABLE)

A-is-on-B AND
B-is-on-C AND
C-is-on-Table

(NOT B-is-on-Table) AND
(NOT C-is-on-A) AND
(NOT A-is-on-Table)

Task Specification

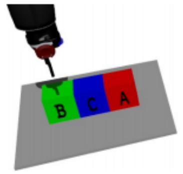
- Task operators
  - State: defined with logical formulas
  - Actions (operators): define changes in state, e.g., `unstack(?x, ?y)`
- Example:
  - Unstacking blocks



On(B,Table) ∧
On(C,A) ∧
On(A,Table) ∧
Clear(B) ∧
Clear(C)

$\xrightarrow{\texttt{Unstack(C,A)}}$

On(B,Table) ∧
On(C,Table) ∧
On(A,Table) ∧
Clear(A) ∧
Clear(B) ∧
Clear(C)



Beyond Propositional Logic

- Propositional logic is lacking

- Consider the following syllogism

1. All men are mortal
2. Socrates is a man

---

3. Therefore, Socrates is mortal

This argument is valid, but it relies on a quantifier "*All*" that cannot be expressed in propositional logic

First-Order Logic

- First-order logic:
  - Propositional Logic + Functions, Relations, and Quantifiers
  - Functions: $2^{var} \rightarrow var$
  - Relations: $2^{var} \rightarrow \{0,1\}$
  - Quantifiers: $\exists, \forall$
- Example:
  - Who is Socrates' father? Father(x)
  - Is Socrates a man? Man(x)

First-Order Logic

- First-order logic:
  - Propositional Logic + Functions, Relations, and Quantifiers
  - Functions: $2^{var} \rightarrow var$
  - Relations: $2^{var} \rightarrow \{0,1\}$
  - Quantifiers: $\exists, \forall$

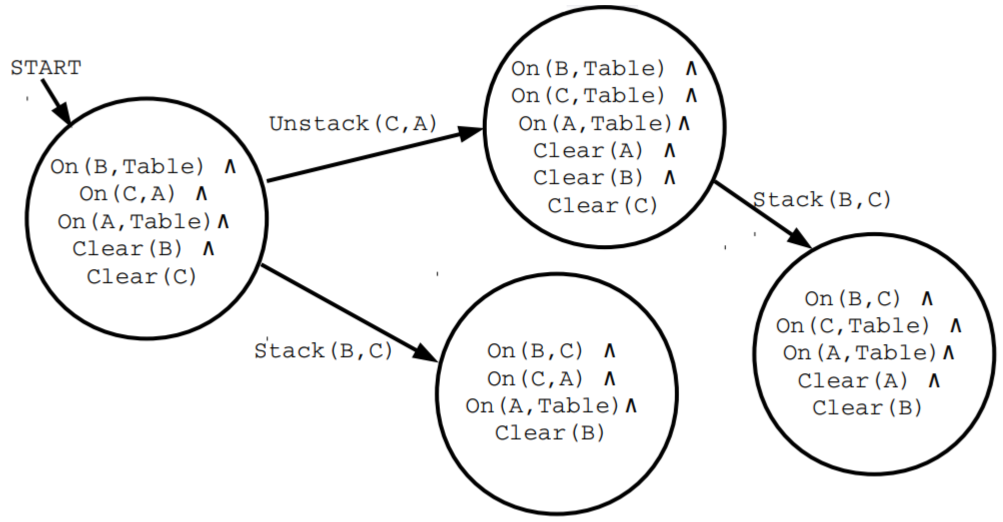
- Example:
  - Who is Socrates' father? Father(x) (function)
  - Is Socrates a man? Man(x) (relation)

First-Order Logic

- First-order logic is undecidable
- We use it to express things succinctly, but we are only interested in a subset:
  - Functions:
    - Usually a constant like `block_a` or `robot_left_hand`
  - Relations:
    - Something like `onTable(?x)` ('?' denotes variables)
  - Quantifiers:
    - $\exists, \forall$
- Reduce some instances to propositional logic:
  - Ground the functions

Task Transitions

- Actions enable transitions:



Planning Domain Definition Language

- Planning Domain Definition Language (PDDL) is a way to express a task planning problem
- Use prefix notation:
  - `(+ 1 2)` returns `3;` `(+ 1 2 (* 3 4))` returns `15`
- Define:
  - Predicates: `on(?x,?y),` `clear(?x),` etc.
  - Actions: `stack(?x,?y),` etc.
  - Initial state: `(ontable a)`, etc.
  - Goal state: `(on a b),(on b c),(ontable a)`

Planning Domain Definition Language

Operators

```
(define (domain blocks)
  (:predicates (on ?x ?y) (ontable ?x)
               (clear ?x) (handempty) (holding ?x))
  (:action pick-up  :parameters (?x)
           :precondition (and (clear ?x) (ontable ?x)
                              (handempty))
           :effect (and (not (ontable ?x))(not (clear ?x))
                        (not (handempty)) (holding ?x)))
  (:action put-down  :parameters (?x)
           :precondition (holding ?x)
           :effect (and (not (holding ?x)) (clear ?x)
                        (handempty)(ontable ?x)))
  (:action stack  :parameters (?x ?y)
           :precondition (and (holding ?x) (clear ?y))
           :effect (and (not (holding ?x))(not (clear ?y))
                        (clear ?x)(handempty) (on ?x ?y)))
  (:action unstack  :parameters (?x ?y)
           :precondition (and (on ?x ?y) (clear ?x)
                              (handempty))
           :effect (and (holding ?x) (clear ?y)
                        (not (clear ?x))
                        (not (handempty))
                        (not (on ?x ?y)))))
```

Facts

```
(define
 (problem sussman-anomaly)
  (:domain blocks)
  (:objects a b c)
  (:init (on c a)
         (ontable a)
         (ontable b)
         (clear c)
         (clear b)
         (handempty))
  (:goal (and (on b c)
              (on a b))))
```

Grounding First-Order Logic

- Rely on the fact that there are **finite**:
  - Constants
  - Objects,
  - locations,
  - steps

- Construct a list of things that could be true or false for each time step
  - e.g.,
    - At(Obj?, Loc?) → PLATE_2_AT_LOC_3
    - At(Obj?, Loc?) → PLATE_1_AT_LOC_5
    - Etc.

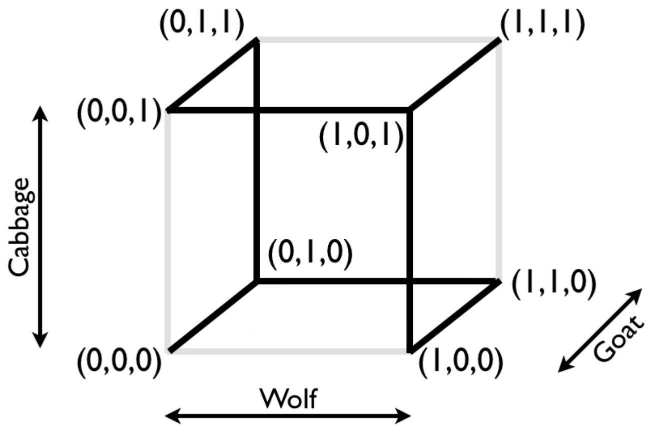
Task Planning Problem with 1st-Order Logic

- **Given**:
  - First-Order Logic Relations
  - First-Order Logic Functions
  - Actions (with preconditions and effects)
  - Start State
  - Goal State(s)

- **Find**:
  - Sequence of actions from start to goal

Approach

- Formal logics allows a way to describe states and transitions
- Use these to build a graph
- Apply standard graph search algorithms in the state space



Approach

- Search state space to find a sequence of actions from start to goal
- Find successor states by looking at all actions that can be applied from a given state
    - this means preconditions are met
- Search until we reach the goal:
    - this search space is typically very large
    - general heuristic search for task planning is an open research problem
    - use heuristic search
        - e.g., A*, Dijkstra's, FF, ...

Summary

- Classical planning, a discrete problem with many applications
- Representation of planning problem
  - State space
  - Initial state
  - Goal state
  - Transitions
- Solving planning problems
  - Heuristic search, e.g., A*
  - General heuristics for task planning is an open research problem.
    - ➢ See International Planning Competition for the latest planners.

References

Book:

- Lavalle: Planning Algorithms
- Chapter: 2

Stuart Russel and Peter Norvig, “Artificial Intelligence, A Modern Approach” 3rd Edition

- Chapters 7, 10, and 11