Intuitions on Planning

- Intelligent Agents: Natural or artificial systems which act in an intelligent way
- Intelligent action is rational action, that is, the best possible action in a given situation
- Planning is the reasoning side of acting
- Abstract, explicit deliberation process that chooses and organizes actions by anticipating their expected outcomes
- Some actions require planning, many do not
  - we act more frequently than we explicitly plan
  - performing well-trained behaviors for which we have pre-stored plans
  - acting and adapting in flexible settings
Intuitions on Planning

- Planning is a complicated, time consuming, and costly process.
- Planning is needed when:
  - new situations, unfamiliar actions are involved
  - complex tasks, complex objectives are addressed
  - actions are constrained by high risks, high costs, joint activities, need for synchronization
- Typically we seek feasible, good plans, not optimal plans (cf. Simon’s “bounded rationality”)
Motivations for Automated Planning

- Practical
  - Designing information processing tools that give access to affordable and efficient planning resources
  - Some professionals face complex changing tasks that involve demanding safety and/or efficiency requirements
  - Example: disaster rescue operations
    - large number of actors, deployment of communication and transportation infrastructure, time constrained, demands for immediate decisions
    - relies on careful planning and assessment of several alternate plans
  - Example: organizers of social meetings
Motivations for Automated Planning cont.

- **Theoretical**
  - Planning is an important component of rational behavior
  - Purpose of AI: grasping computational aspects of intelligence
  - planning, as the reasoning side of acting, is a key element
  - Studying planning as abstract process (complexity, efficiency of algorithms, ...)
  - Planning as integrated component of deliberative behavior
Motivations for Automated Planning cont.

- Hot topic: study and design of autonomous intelligent machines
  - satellites, spacecrafts, robots cannot always be teleoperated
  - interaction with nonexpert humans on task level rather than control signals
  - machines that can sense and act as well as reason on their actions
Automated Planning

- Plan: Sequence of actions to achieve a goal
- Planning: Computation of such a sequence
- Examples of successful applications
  - Space Exploration
  - Manufacturing
  - Games
Space Exploration

- Autonomous planning, scheduling, control
  - NASA: JPL and Ames
- Remote Agent Experiment (RAX)
  - Deep Space 1
- Mars Exploration Rover (MER)
Manufacturing

- Sheet-metal bending machines - Amada Corporation
  - Software to plan the sequence of bends
    [Gupta and Bourne, *J. Manufacturing Sci. and Engr.*, 1999]
Games

- *Bridge Baron* - Great Game Products
  - 1997 world champion of computer bridge
    [Smith, Nau, and Throop, *AI Magazine*, 1998]
  - 2004: 2nd place
Conceptual Model of Planning

1. Environment

- Initial state
- Objectives
- Execution status
- Plans
- Observations
- Actions
- Events

System $\Sigma$

State transition system $\Sigma = (S, A, E, \gamma)$

- $S$ = {states}
- $A$ = {actions}
- $E$ = {exogenous events}
- $\gamma$ = state-transition function
Conceptual Model of Planning

State Transition System

\[ \Sigma = (S, A, E, \gamma) \]

- \( S = \{ \text{states} \} \)
- \( A = \{ \text{actions} \} \)
- \( E = \{ \text{exogenous events} \} \)
- State-transition function \( \gamma: S \times (A \cup E) \rightarrow 2^S \)

- \( S = \{ s_0, \ldots, s_5 \} \)
- \( A = \{ \text{move1, move2, put, take, load, unload} \} \)
- \( E = \{ \} \)
- \( \gamma: \) see the arrows

The Dock Worker Robots (DWR) domain
Conceptual Model of Planning

2. Controller

- Initial state
- Objectives
- Execution status

Planner

- Description of $\Sigma$
- Plans
- Observations
- Actions

Controller

System $\Sigma$

Observation function $h: S \rightarrow O$

Given observation $o$ in $O$, produces action $a$ in $A$
Conceptual Model of Planning

3. Planner’s Input

- Planning problem
- Initial state
- Objectives
- Execution status
- Descriptions of $\Sigma$
- Plans
- Controller
- Observations
- Actions
- System $\Sigma$
- Events

Omit unless planning is online
Conceptual Model of Planning

Planning Problem

Description of $\Sigma$
Initial state or set of states
   Initial state $= s_0$
Objective
   Goal state, set of goal states, set of tasks, “trajectory” of states, objective function, …
   Goal state $= s_5$

The Dock Worker Robots (DWR) domain
Conceptual Model of Planning

**Conceptual Model**

4. Planner’s Output

- **Initial state**
- **Objectives**
- **Execution status**
- **Plan**
- **System Σ**
- **Observations**
- **Actions**
- **Events**

- **Description of Σ**

---

- **Planner**

---

- **Controller**

---

- **Instructions to the controller**
Conceptual Model of Planning

**Plans**

**Classical plan**: a sequence of actions

\(<\text{take}, \text{move1}, \text{load}, \text{move2} >\)

**Policy**: partial function from \(S\) into \(A\)

\[\{ (s_0, \text{take}), (s_1, \text{move1}), (s_3, \text{load}), (s_4, \text{move2}) \}\]

The Dock Worker Robots (DWR) domain
Syntax of FOL Terms and Formulae

Inductive definition:

- **Terms:**
  - A variable \( v \in V \) is a term.
  - If \( f \) is a function symbol with arity \( n \) and \( t_1 \ldots t_n \) are terms, then
    \( f(t_1, \ldots, t_n) \) is a term. (including constant symbols as 0-ary function symbols)
  - That are all terms.

- **Formulas:**
  - if \( P \) is a predicate symbol with arity \( n \) and \( t_1 \ldots t_n \) are terms, then
    \( P(t_1, \ldots, t_n) \) is a formula. (atomic formula)
  - For all formulas \( F \) and \( G \), \( \neg F \), \( F \land G \), \( F \lor G \), \( F \rightarrow G \) and \( F \leftrightarrow G \) are formula. (connectives “not”, “and”, “or”, “implies”, “equivalent”)
  - If \( v \) is a variable and \( F \) is a formula, then \( \exists v \ F \) and \( \forall v \ F \) are formulas. (existential and universal quantifier, “exists”, “for all”)
  - That are all formulas.
Remarks on Syntax of FOL

- Formula are constructed over terms. Never confuse this categories!
- Additionally, parentheses can be used to group sub-expressions.
- Expressions which obey the given inductive definition are called well-formed formulas (wwfs).
  The closure “that are all terms/formulas” is necessary to exclude all other kinds of (not well-formed) expressions.
- We refer to atomic formulas also as “atoms”. Positive and negated atoms \((P, \neg P)\) are called positive/negative literals.
Remarks on Syntax of FOL cont.

- A variable which is in the scope of a quantor is called **bound**, otherwise it is called **free**.
  
  \[ P(x) \lor \forall y \exists z \ Q(y, z) \]
  
  `x` is free and `y` and `z` are bound.

- A formula without free variables is called **sentence**.

- Propositional logic is a special case of FOL: use only unary predicate symbols (then there are no terms, no variable and no quantors) or just forbid variables and quantors (use only grounded formulas).
Problem solving: using domain-specific heuristics to search for a (optimal) sequence of actions

Scheduling: decide when and how to perform a given set of actions obeying time constraints, resource constraints, objective functions

Planning: decide what actions to use in what sequence to achieve some set of objectives

biggest algorithmical challenge: often worse than NP-complete, worst case is undecidable (see lecture about complexity of classical planning)
Domain-Independent Planning

- In principle, a domain-independent planner works in any planning domain.
- Uses no domain-specific knowledge except the definition of the basic actions.
- In practice, it is not feasible to develop a planner that works in every possible domain.
- Make simplifying assumptions to restrict the set of domains: mostly classical planning.
- Domain-specific planners can be very successful in specific domains but one needs to write an entire program (lots of work).
Classical Planning

- Restrictive assumptions (see next slide): finite set of states and actions; fully observable states; deterministic outcome of actions, ...
- Reduces to the problem of path searching in a graph with nodes as states and edges as actions (which is still hard enough):
  - Generalize the earlier example to 5 locations, 3 robot carts, 100 containers, 3 piles: $10^{277}$ states
  - Number of particles in the universe is about $10^{87}$
- Most research is on classical planning with many different algorithms
- Planning Competition (AIPS 1998, AIPS 2000, IPC 2002, ...) shows the progress every two years
Restrictions

restrictive Assumptions

- **A0**: Finite system:
  - finitely many states, actions, events
- **A1**: Fully observable:
  - the controller always $\Sigma$'s current state
- **A2**: Deterministic:
  - each action has only one outcome
- **A3**: Static (no exogenous events):
  - no changes but the controller’s actions
- **A4**: Attainment goals:
  - a set of goal states $S_g$
- **A5**: Sequential plans:
  - a plan is a linearly ordered sequence of actions $(a_1, a_2, \ldots, a_n)$
- **A6**: Implicit time:
  - no time durations; linear sequence of instantaneous states
- **A7**: Off-line planning:
  - planner doesn’t know the execution status
A running example: Dock Worker Robots

- Generalization of the earlier example
  - A harbor with several locations
    - e.g., docks, docked ships, storage areas, parking areas
  - Containers
    - going to/from ships
  - Robot carts
    - can move containers
  - Cranes
    - can load and unload containers
A running example: Dock Worker Robots

- **Locations:** l1, l2, ...
- **Containers:** c1, c2, ...
  - can be stacked in piles, loaded onto robots, or held by cranes
- **Piles:** p1, p2, ...
  - fixed areas where containers are stacked
  - pallet at the bottom of each pile
- **Robot carts:** r1, r2, ...
  - can move to adjacent locations
  - carry at most one container
- **Cranes:** k1, k2, ...
  - each belongs to a single location
  - move containers between piles and robots
  - if there is a pile at a location, there must also be a crane there
A running example: Dock Worker Robots

- Fixed relations: same in all states
  - \texttt{adjacent}(l,l')
  - \texttt{attached}(p, l)
  - \texttt{belong}(k, l)

- Dynamic relations: differ from one state to another
  - \texttt{occupied}(l)
  - \texttt{at}(r, l)
  - \texttt{loaded}(r, c)
  - \texttt{unloaded}(r)
  - \texttt{holding}(k, c)
  - \texttt{empty}(k)
  - \texttt{in}(c, p)
  - \texttt{on}(c, c')
  - \texttt{top}(c, p)
  - \texttt{top}(pallet, p)

- Actions:
  - \texttt{take}(c, k, p)
  - \texttt{put}(c, k, p)
  - \texttt{load}(r, c, k)
  - \texttt{unload}(r)
  - \texttt{move}(r, l, l')
Closed-World Assumption (CWA)

- An atom that is not explicitly given in a state does not hold in the state
- That is: Assumption of the value \textit{false} for every atom which is not explicitly stated
- Classical, set-theoretical and state-variable representation all rely on the CWA
- CWA is a restriction of the logic calculus: no true negation but \textit{negation by failure} (If a proposition cannot be proven to be true, it is assumed to be false.)
This restriction makes state-based planning more efficient than deductive planning in full FOL where the frame problem exists.

Frame problem: not only the propositions which change by an action must be specified but also all propositions which are not affected by an action (e.g. If I put block $x$ from block $y$ on the table, $on(y,z)$ still holds).
Extended Representation

- Typed variables and relations
- Conditional Operators
- Quantified Expressions
- Equality Constraints
- Disjunctive Preconditions
- Function Symbols
- Axiomatic Inference
- Attached Procedures
Problem Domain Definition Language as common language for most modern planners (see PDDL Specification)

Example: Equality constraints and conditioned effects

(define (domain blocksworld-adl)
 (:requirements :strips :equality :conditional-effects)
 (:predicates (on ?x ?y)
     (clear ?x)) ; clear(Table) is static
 (:action puton
  :parameters (?x ?y ?z)
  :precondition (and (on ?x ?z) (clear ?x) (clear ?y)
                  (not (= ?y ?z)) (not (= ?x ?z))
                  (not (= ?x ?y)) (not (= ?x Table)))
  :effect (and (on ?x ?y) (not (on ?x ?z))
               (when (not (eq ?z Table)) (clear ?z))
               (when (not (eq ?y Table)) (not (clear ?y))))))
)