Intelligent Agents

Introduction to Planning

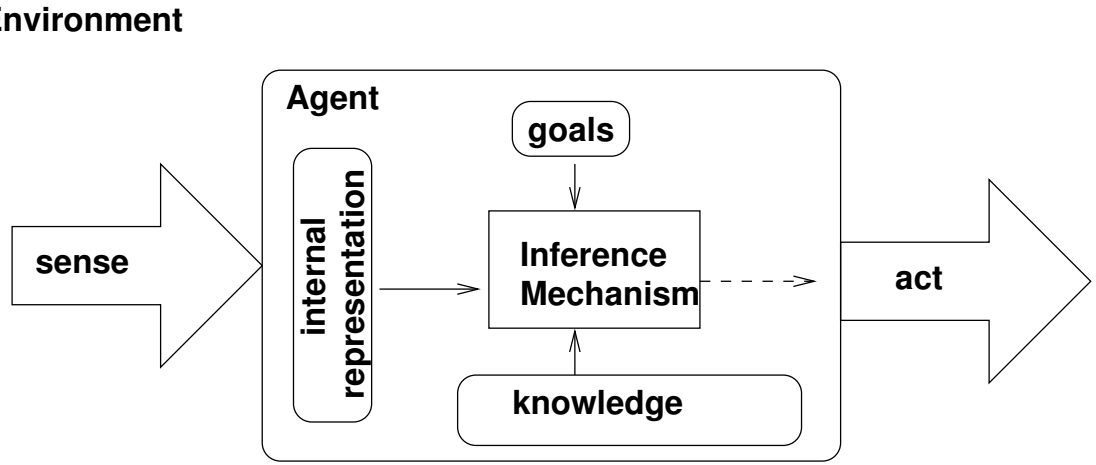
Ute Schmid

Cognitive Systems, Applied Computer Science, Bamberg University

slides partially by **Dana S. Nau**
CMSC 722, AI Planning
University of Maryland, Spring 2008

last change: 27. April 2015

Intelligent Agent



- Embedded in an environment (sensing and acting)
- Inference mechanism
  - based on internal representation of the perceived information
  - usage of knowledge (facts, rules)
  - maybe goal-directed
- Examples for inference mechanisms:
  - theorem proving
  - **planning**

Inference Mechanism

- An algorithmic method which workes on a defined representation language
- On the machine:
  - correct syntactical expressions of a language
  - complete and correct procedure (returns a result if one existis and a result is correct wrt the intended semantics)
- Semantics is outside of the machine, can be related to the defined procedure
- e.g., resolution calculus: a syntactic procedure for automated theorem procing with a defined semantics for first-order logic

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

- *Plan*: Sequence of actions to achieve a goal
- *Planning*: Computation of such a sequence

- 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")

Motivation 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*
    time constraints, maybe child care facility necessary, different nutrition requirements, presentation equipment

Motivation for Automated Planning cont.

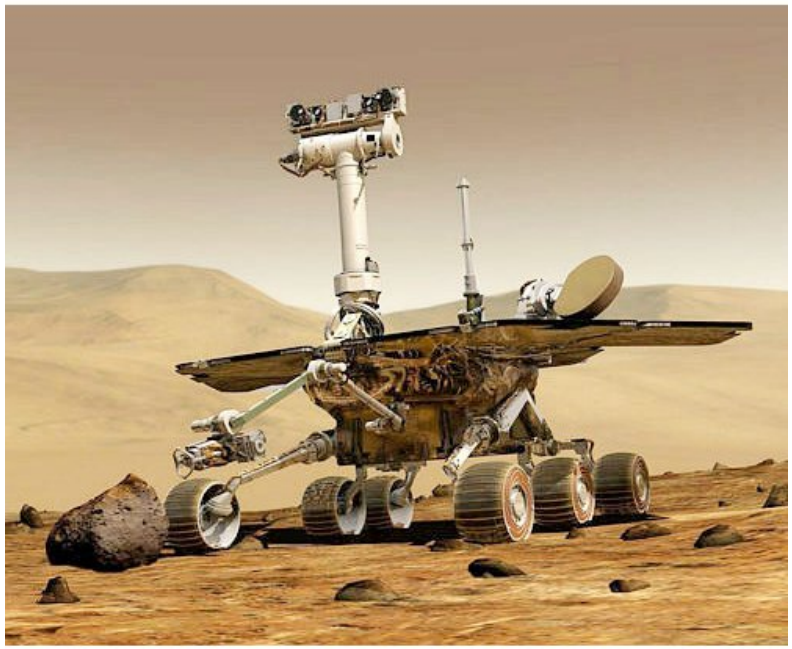
- Theoretical
  - Planning is an important component of rational behavior
  - Purpose of AI: grasping computational aspects of intelligence
  - Planning, as the reasoning side of acting, is a key element
  - Studying planning as abstract process (complexity, efficiency of algorithms, ...)
  - Planning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

Motivation for Automated Planning cont.

- **Current Research:** study and design of **autonomous** intelligent machines
  - satellites, space-crafts, robots cannot always be tele-operated
  - interaction with non-expert humans on task level rather than control signals
  - machines that can sense and act as well as reason on their actions
- Examples of successful applications
  - Space Exploration
  - Manufacturing
  - Games

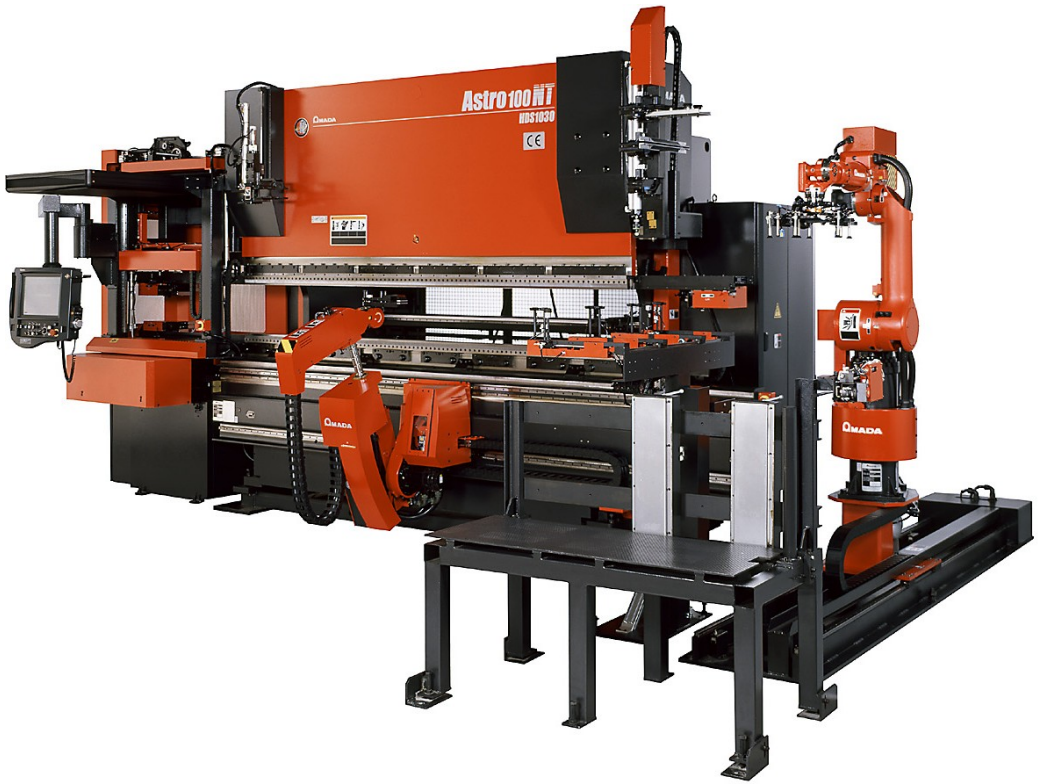
Space Exploration

- Autonomous planning, schedule, control
  - NASA: JPL and AMES
- Remote Agent Experient (RAX)
  - Deep Space 1
- Mars Exploration Rover (MER)



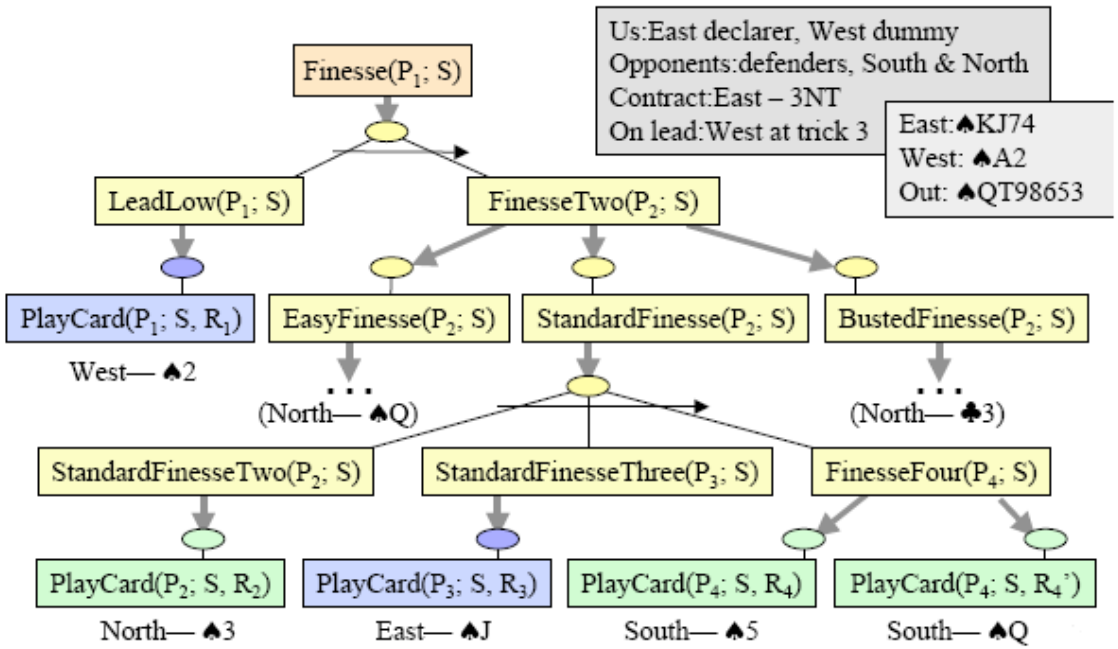
Manufacturing

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



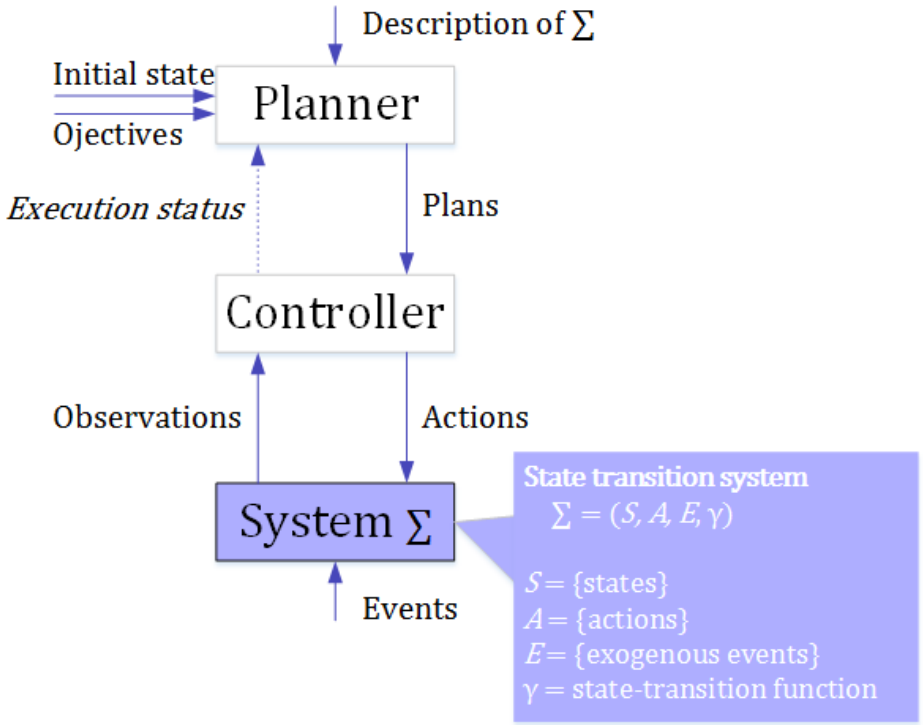
AI in Games

- *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

Conceptual Model
1. Environment

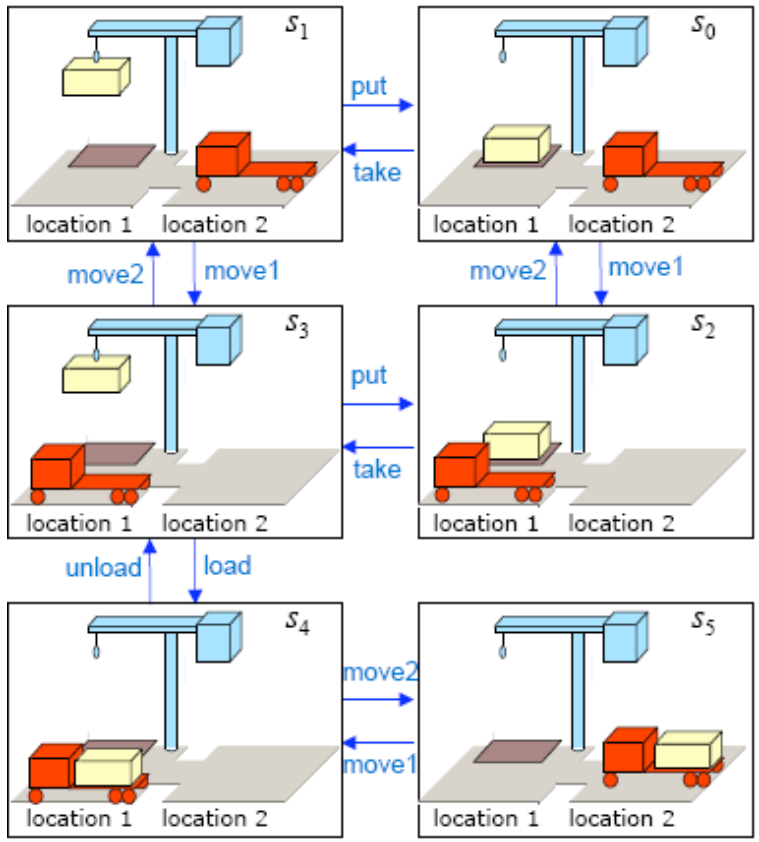


Conceptual Model of Planning

State Transition System

$\Sigma = (S, A, E, \gamma)$

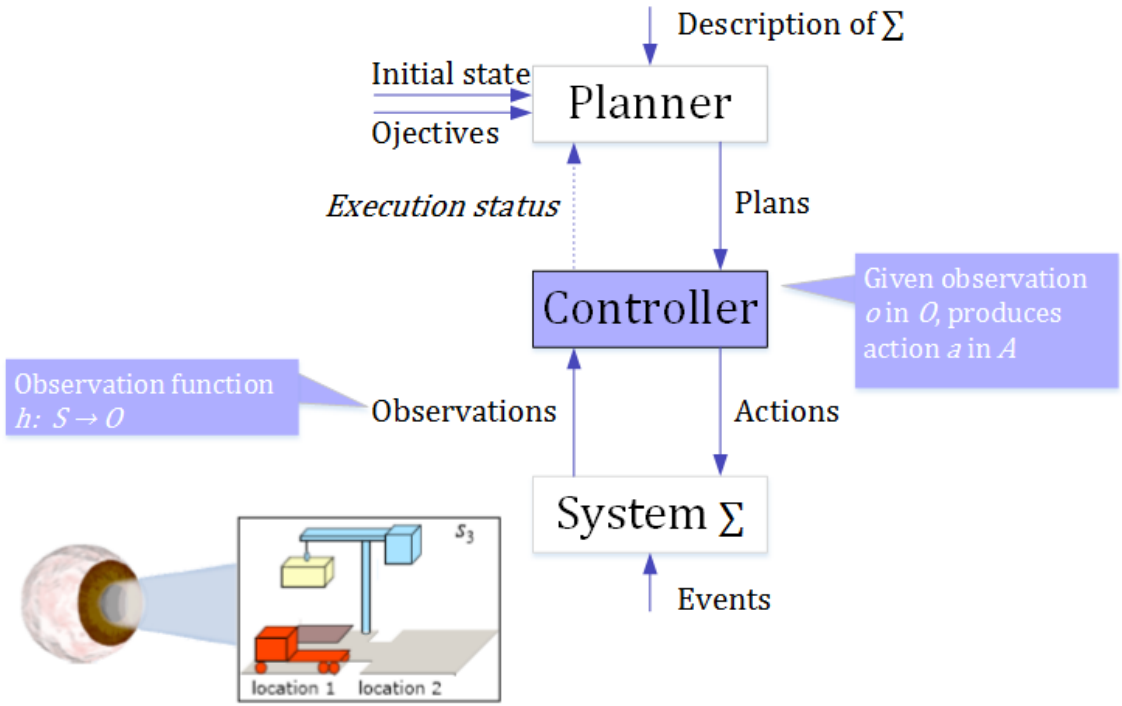
- $S = \{states\}$
- $A = \{actions\}$
- $E = \{exogenous\ events\}$
- State-transition function
  $\gamma : S \times (A \cup E) \rightarrow 2^S$
  - $S = \{s_0, ..., s_5\}$
  - $A = \{move1, move2, put, take, load, unload\}$
  - $E = \{\}$
  - γ : see the arrows



The Dock Worker Robots (DWR) domain

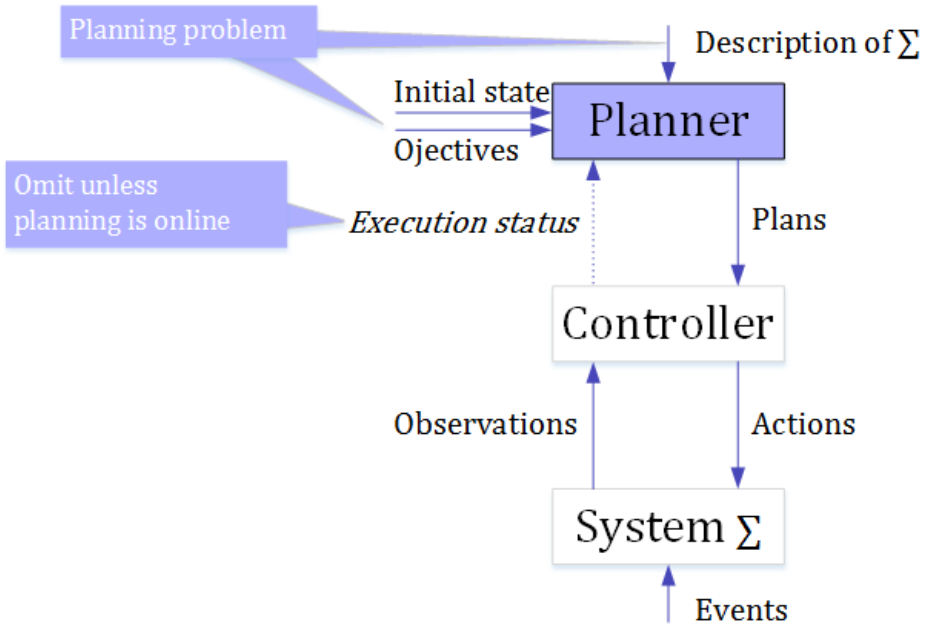
Conceptual Model of Planning

Conceptual Model
2. Controller



Conceptual Model of Planning

Conceptual Model
3. Planner's Input



Conceptual Model of Planning

Planning Problem

Description of Σ

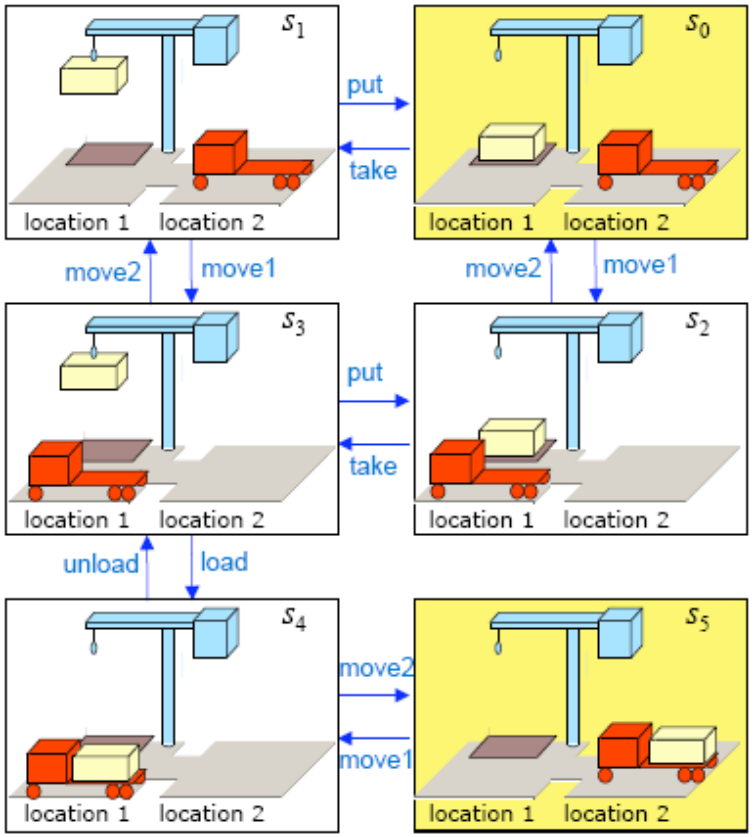
Initial state or set of states

Initial state $= s_0$

Objective

Goal state, set of goal states, set of tasks, "trajectory" of states, objective function, ...

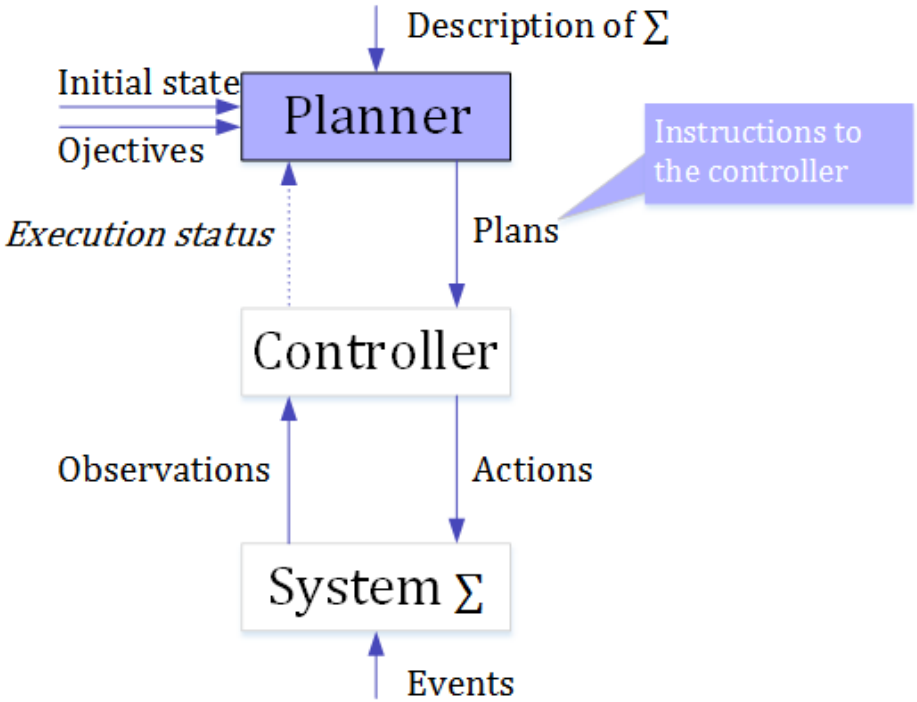
Goal state $= s_5$



The Dock Worker Robots (DWR) domain

Conceptual Model of Planning

Conceptual Model
4. Planner's Output



Conceptual Model of Planning

Plans

Classical plan:

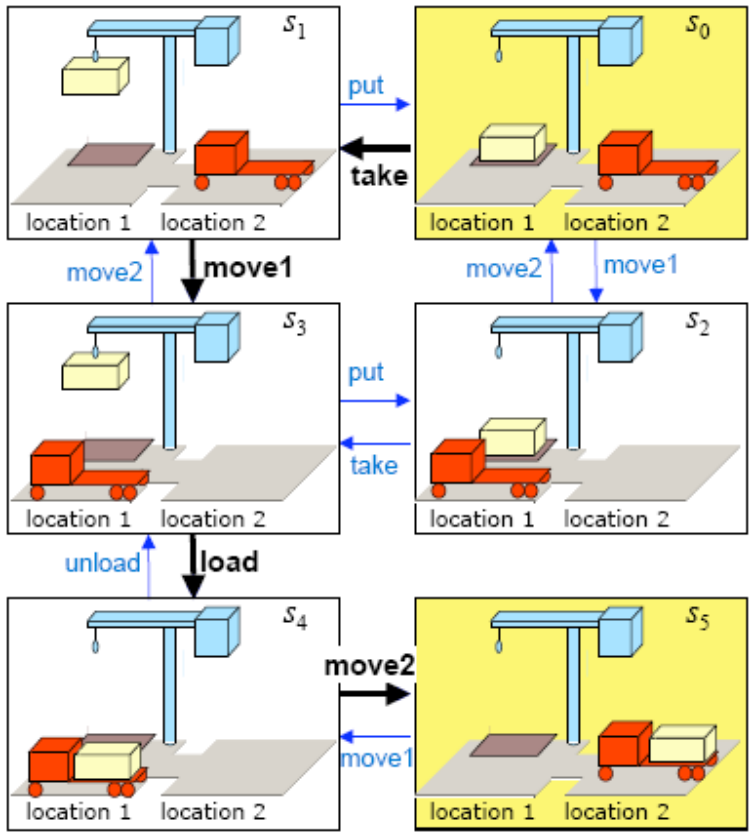
a sequence of actions

$\langle take, move1, load, move2 \rangle$

Policy:

partial function from S into A

$\{(s_0, take),$
$(s_1, move1),$
$(s_3, load),$
$(s_4, moves2)\}$

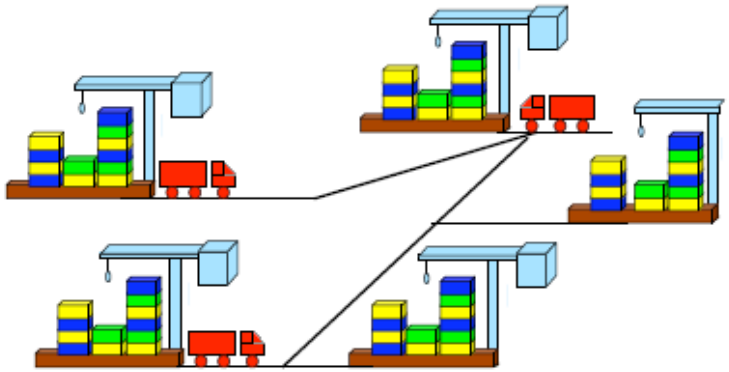


The Dock Worker Robots (DWR) domain

Running Example

A running example: Dock Worker Robots

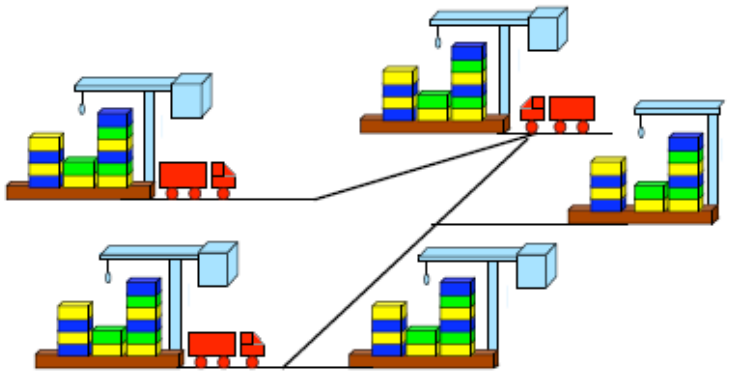
- Generalization of the earlier example
  - A harbor with several locations
    - ↪ e. g., docks, docked ships, storage areas, parking areas
  - Containers
    - ↪ going to/from ships
  - Robot carts
    - ↪ can move containers
  - Cranes
    - ↪ can load and unload containers



Running Example

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



Running Example

A running example: Dock Worker Robots

- Fixed relations:

  same in all states

  adjacent(*l,l'*) attached(*p,l*) belong(*k,l*)

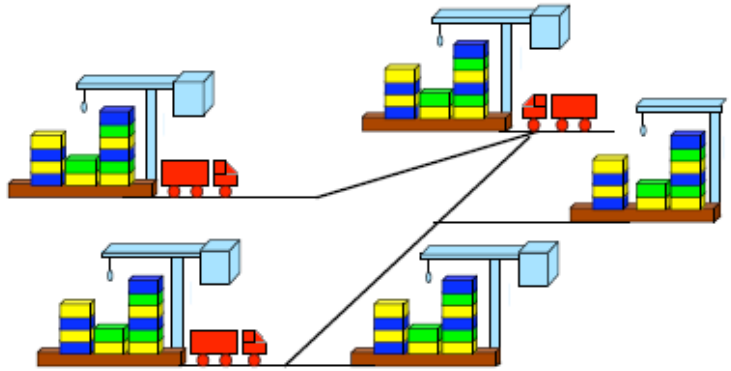
- Dynamic relations:

  differ from one state to another

  occupied(*l*) at(*r,l*)
  loaded(*r,c*) unloaded(*r*)
  holding(*k,c*) empty(*k*)
  in(*c,p*) on(*c,c'*)
  top(*c,p*) top(*pallet,p*)

- Actions:

  take(*c,k,p*) put(*c,k,p*)
  load(*r,c,k*) unloaded(*r*) move(*r,l,l'*)



Discussion of Important Concepts

- Environment: Compare to what you learned in logic
- Description of Σ (Syntax) vs. Environment Σ (Semantics)
- Problem Domain and Problem
- Domain-independent vs. domain-dependent planning
- Closed-world assumption (CWA)
- Planner: An inference mechanism which computes a plan from a domain and a problem description
- Three related domains: Planning, problem solving, scheduling
- Classical planning and its restrictive assumptions
- Representation language PDDL

Environment and What You Know From Logic

- We introduced the environment formally as a state-transition system Σ
- The environment represents the *world* in which the action sequence calculated by a planner from a *representation of the world* has to be feasible:
  - Calculating an action as transformation from one state representation to another state representation has to guarantee that the calculated succesor representation corresponds to a situation in the world which can be obtained from the predecessor situation by applying the action!
  - A plan representing a sequence of actions has to correspond to a problem solution in the world!
- On the following slides: a repetition from what you learned in your logic course
- Be aware that in logic you used Σ to represent a signature of a formal language, that is, in this context Σ refers to a representation (syntax)

More Formal Treatment of Representation

First-order language, function-free and without equality:
Signature $\Sigma = (S, OP, REL)$

- *Do not confuse signature "Sigma" with state-transition system (also denoted with Σ)!*
- *Do not confure sorts S with the set of states in the state-transition system!*

- A signature defines the syntactical structure of a language
- It is defined by three sets: sorts S, operators OP, and relations REL
- Operators have different arities, constants are operators with arity zero (range represents "type")
- Relations have range "prop" which can be true or false
- Never confuse operators (with a range of some sort) and relations which result in a truth value!

as introduced by Michael Mender

More Formal Treatment of Representation

Sorts S:

$$S = \{ \quad \textit{location, container, pile, } \ldots \}$$

Operators OP:

$$\begin{aligned} OP = \{ \quad & l_1, l_2, \ldots : \text{location}, \\ & c_1, c_2, \ldots : \textit{container}, \\ & p_1, p_2, \ldots : \textit{piles}, \\ & r_1, r_2, \ldots : \textit{robot carts}, \\ & k_1, k_2, \ldots : \textit{cranes} \} \end{aligned}$$

Relations REL:

$$\begin{aligned} REL = \{ \quad & \textit{adjacent: location} \times \textit{location} \rightarrow \textit{Prop}, \\ & \textit{occupied: location} \rightarrow \textit{Prop}, \\ & \textit{loaded: robot\_cart} \times \textit{container} \rightarrow \textit{Prop}, \\ & \ldots \} \end{aligned}$$

More Formal Treatment of Representation

- **Ground** (or **closed**) expression: contains no variable symbols
  - e.g., in(c1,p3) *formally, we write* $\emptyset \vdash in(c_1, p_3) : Prop$
- **Unground** (or **open**) expression: at least one variable symbol
  - e.g., in(c1,x) *formally, we write* $x : pile \vdash in(c_1, x) : Prop$

- **Substitution**: $\theta = \{x_1 \leftarrow v_1, x_2 \leftarrow v_2, \cdots, x_n \leftarrow v_n\}$
  - Each x_i is a variable symbol; each v_i is a term (of the same type)
  - θ is *ground* if all v_i are ground
- **Instance** $e\theta$ of expression e: $in(c_1, x)\{x \leftarrow p_3\} = in(c_1, p_3)$
  - result of applying a substitution θ to e
  - Replace variables of e *simultaneously*, not sequentially
  - if $e\theta$ is ground, called *ground* instance.

$$\frac{\emptyset \vdash p_3 : pile \qquad x : pile \vdash in(c_1, x) : Prop}{\emptyset \vdash in(c_1, x)\{x \leftarrow p_3\} : Prop}$$

as introduced by Michael Mender

State-Space Model, Planning Domain, Planning Problem

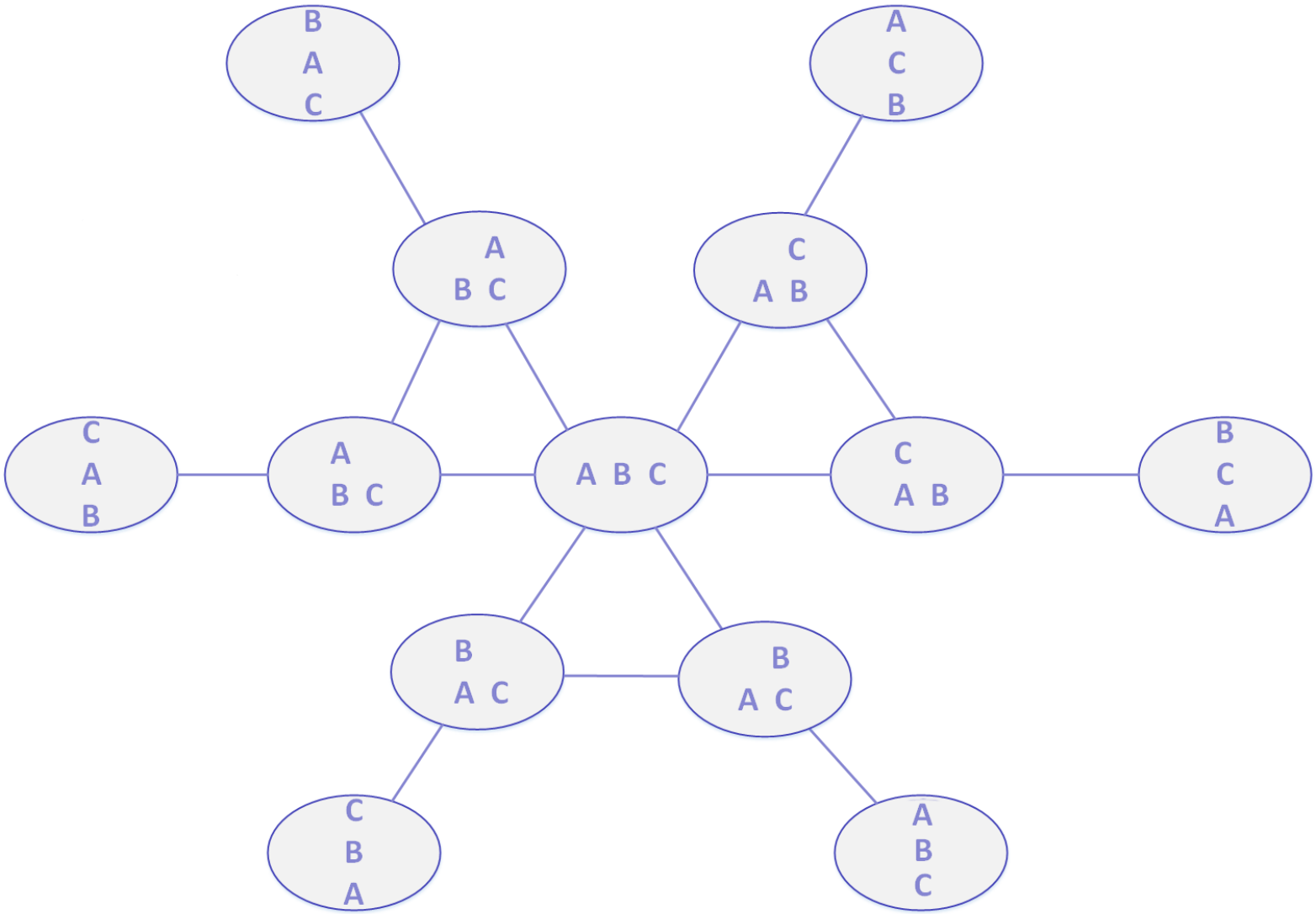
- The environment Σ, i.e., the semantics of a problem domain, is sometimes conceptualized as a state-space model.
- **State-space model:** a graph with all situations (of a finite domain) as nodes and arcs between nodes representing actions which lead from one situation to another. (in cognitive AI also called a problem space – introduced by Newell and Simon)
- **Planning domain:** Σ or its description
- **Planning problem:** the domain together with a specific initial state and objectives (goals)
- Example: Dock-working robots is a domain with an underlying state-space model; for a specific initial state, there exists a corresponding situation in Σ and there exist an arbitrary number of situations which fulfill the objectives.
- The state-space model can be drawn for small problems. It not part of the representation of the planning problem!

State-Space and Search Tree

- State spaces are for most planning problems too large to be represented completely
- Remember: number of states for DWR
- Therefore, standard graph search algorithms (such as the Dijkstra algorithm) for finding the shortest path between two nodes (i.e. initial state and a state fulfilling the goals) cannot be applied
- During search for a plan, a part of the state-space is explored (search tree)

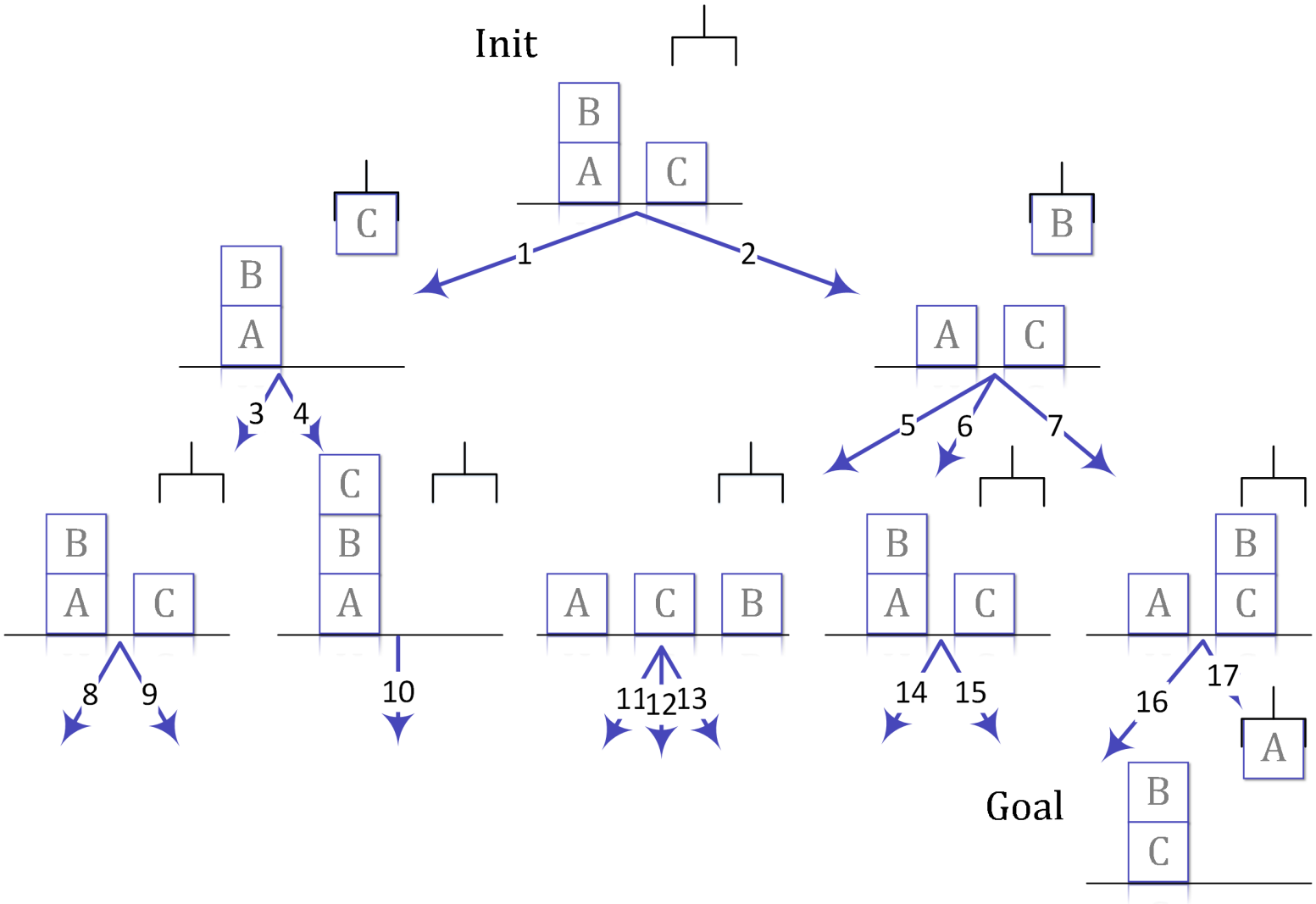
State-Space and Search Tree

- Example: problem space



State-Space and Search Tree

- Example: search tree



Representation of Problem States

- Usually, representation of problem states are based on some restrictive variant of first order logic (FOL).
  {occupied(L1), loaded(R1,C1), on(C2,C3), in(C2,P1), ...}
- A state representation is a set of predicate symbols which represent relations which hold in the situation.
- Typically, arguments of predicate symbols are restricted to constants and variables.
- A single predicate symbol is called **atom**.

- Representations for classical planning will be introduced in a later lecture.
- FOL will also be introduced in a later lecture, in the context of deductive planning.

Representation of Problem States

- In most problems, far too many states to try to represent all of them explicitly as $s_0, s_1, s_2, \ldots$
- Represent each state as a set of features
  - e.g.,
    - $\Rightarrow$ a vector of values for a set of variables
    - $\Rightarrow$ a set of ground atoms in some first-order language L
- Define a set of *operators* that can be used to compute state-transitions
- Don't give all of the states explicitly
  - Just give the initial state
  - Use the operators to generate the other states as needed

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)

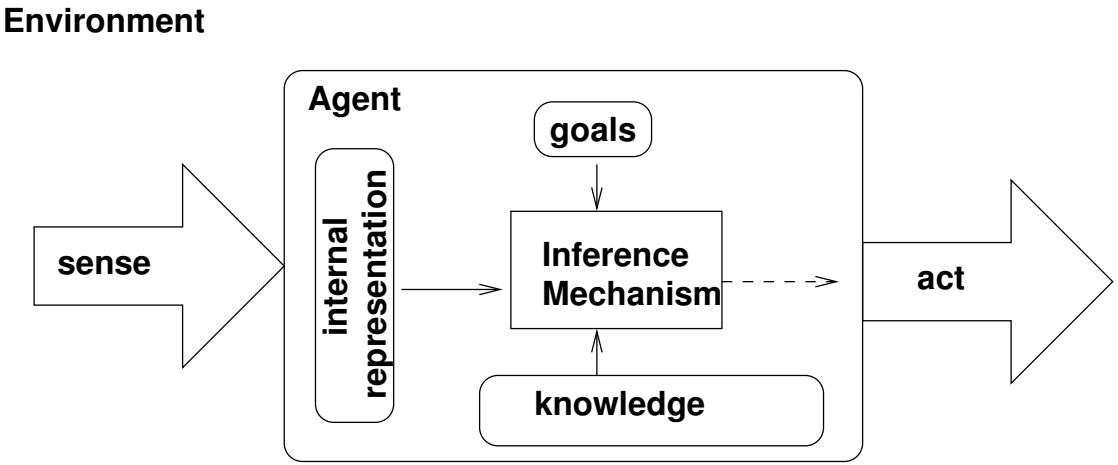
Closed-World Assumption (CWA)

- An atom that is not explicitly given in a state does not hold in the state
- That is: Assumption of the value *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 *negation by failure*
  (If a proposition cannot be proven to be true, it is assumed to be false.)

CWA cont.

- 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)*

Planning for Intelligent Agents



- The inference/planning procdure can be realized in different ways
  - Planning as heuristic search
  - Planning as deductive reasoning
  - Planning as model checking
  - . . .
- Approaches in this course are introduced in the context of planning but are also applied in other areas

Planning, Problem Solving, Scheduling

- 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)

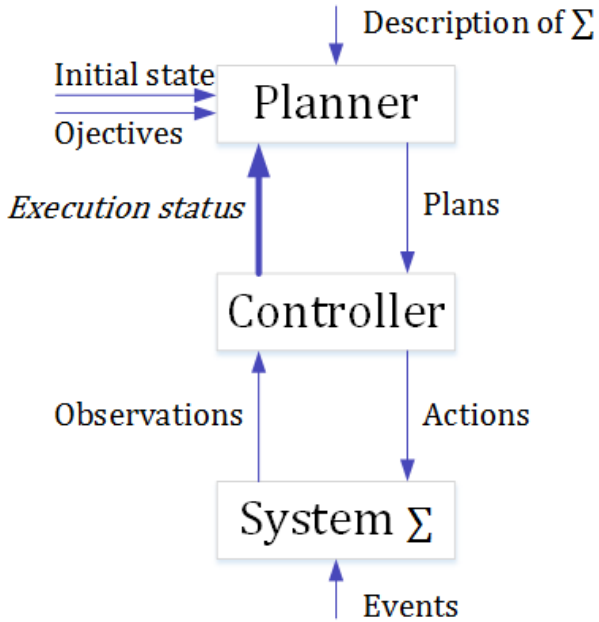
Classical Planning

- 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, now at ICAPS) shows the progress over the years

Restrictions

Restrictive Assumptions

- **A0: Finite system:**
  - ↪ finitely many states, actions, events
- **A1: Fully observable:**
  - ↪ the controller always Σ' 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



Preview: PDDL

- Problem Domain Definition Language as common language for most modern planners (see PDDL Specification)
- Example: Equality constraints and conditioned effects

```
(define (domain blocks-world-domain)
  (:requirements :strips :equality :conditional-effects)
  (:constants Table)
  (:predicates (on ?x ?y)
               (clear ?x)
               (block ?b)
               )
  ;; Define step for placing one block on another.
  (: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 (= ?Z Table)) (clear ?Z))
                  (when (not (= ?Y Table)) (not (clear ?Y))))))
```

Preview: PDDL

- The domain blocksworld is modeled with the *puton* operator "Put block ?x from entity ?z (a block or the table) on entity ?y (a block or the table)
- Requirements specify which language features a planner dealing with this domain must support
- The operator is specified with application conditions (precondition) and effects

Preview: PDDL

- Precondition:
  If block ?x is on ?z and ?x and ?y are clear (and if no two objects are identical and ?x is of type block) then the operator can be applied
- Effect:
  Putting ?x from ?z on ?y results in a state where ?x is on ?y (addition of atoms which hold after application) and no longer on ?z (deletion of atoms which do no longer hold after application)
- A conditioned effect (when) allows to write operators more compactly (without these conditions, there would be three different operators: putBlockonBlock, putBlockFromTableToBlock, putBlockFromBlockonTable)

Summary

- Planning is the reasoning side of action
- There are many application domain, e.g, in the domain of autonomous robotics, manufacturing, and games
- The conceptual model of planning defines an environment as state-transition system and a planner receives a description of the environment together with a problem statement (initial state and objectives) as input to generate a plan as a sequence of actions
- Planning algorithms cannot work on a state-space. Part of the state-space is constructed during search for a plan
- Dock Worker Robots is introduced as running example
- Many approaches concern domain-independent planning
- For classical planning there are restrictive assumptions
- The closed-world assumption means that if an atom is not explicitly given in a state description, it is assumed to not hold in the state
- An example for a blocksworld-domain represented in PDDL was introduced