Automated Planning & Artificial Intelligence

Introduction to AI Planning

Humbert Fiorino

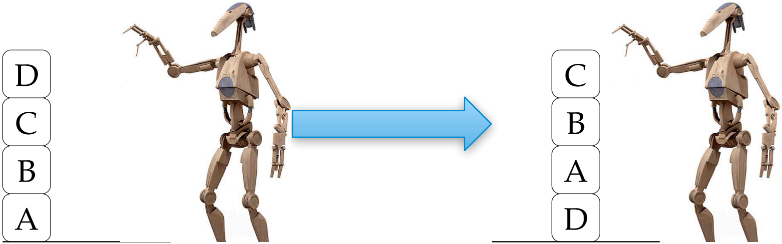
Humbert.Fiorino@imag.fr
http://membres-lig.imag.fr/fiorino

Laboratory of Informatics of Grenoble – MAGMA team

February 2011

Blocks World

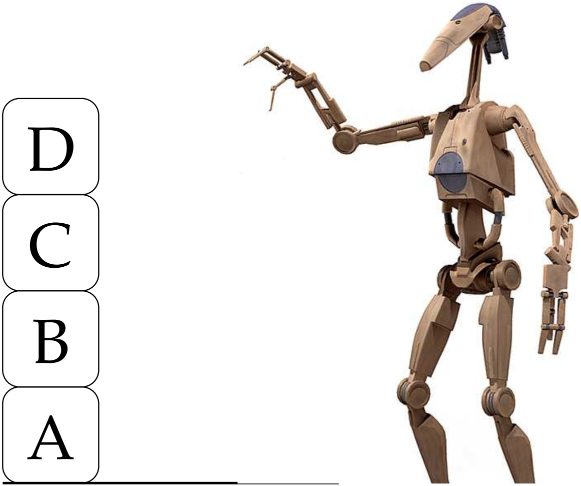
How do we get from the initial state to the goal?



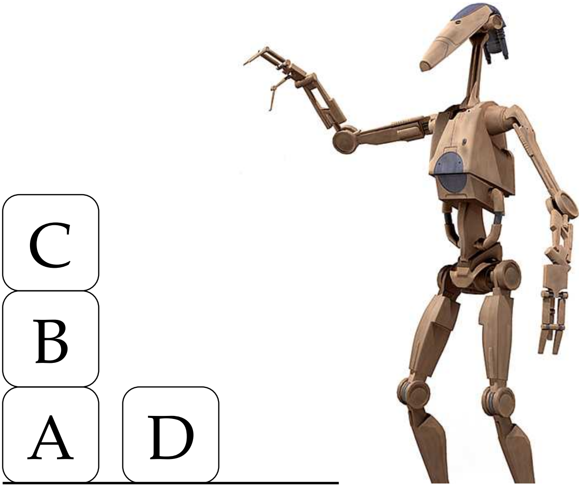
Initial state

Goal

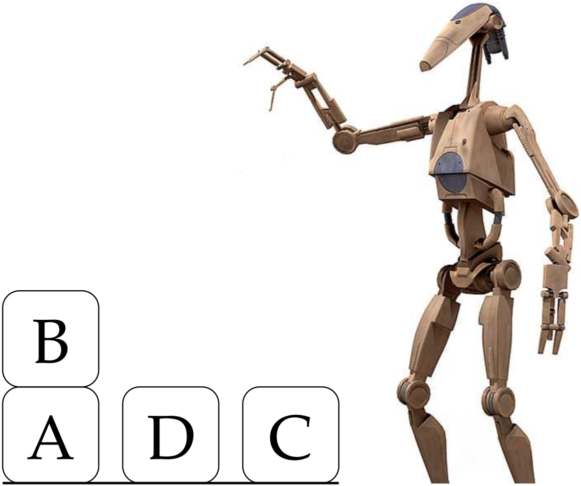
Blocks World



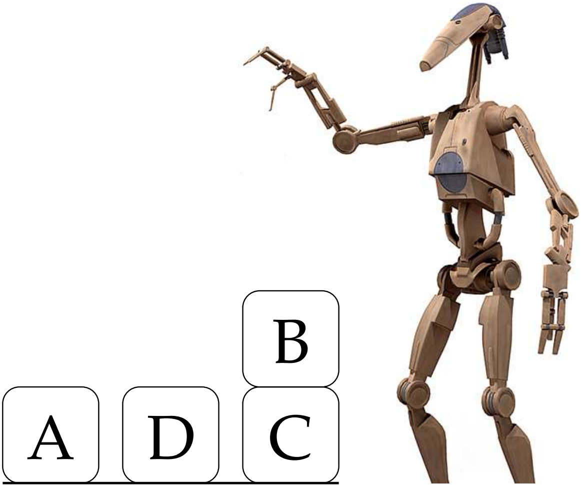
Blocks World



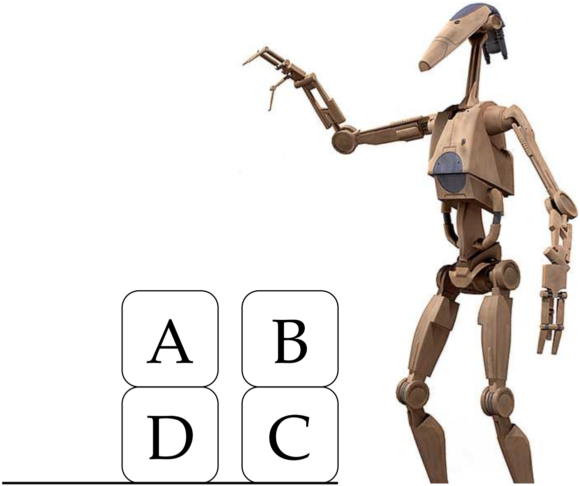
Blocks World



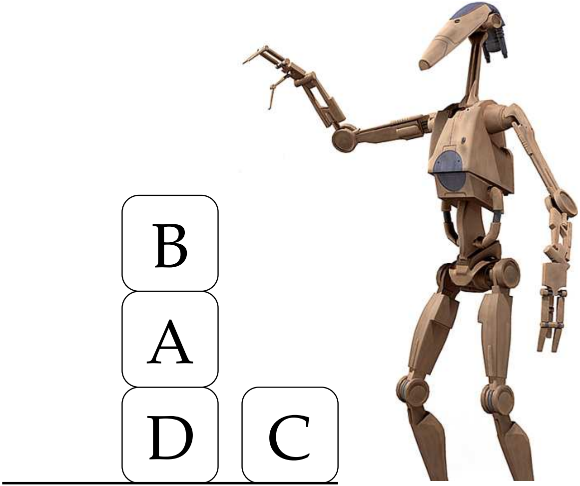
Blocks World



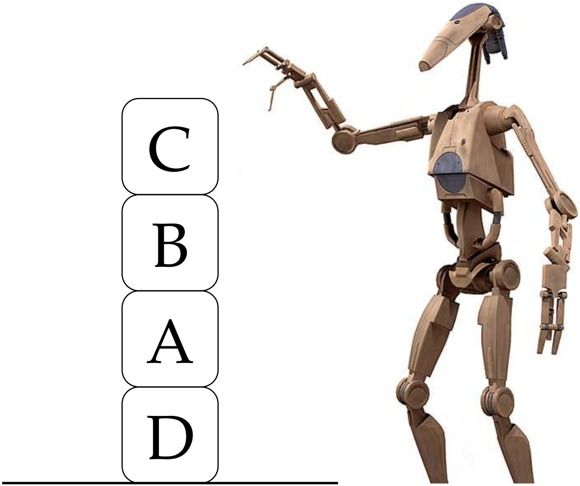
Blocks World



Blocks World

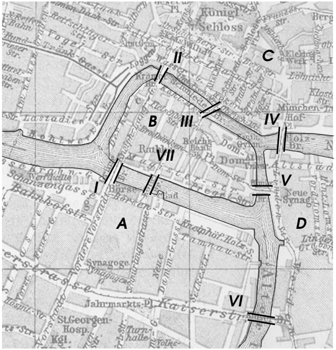


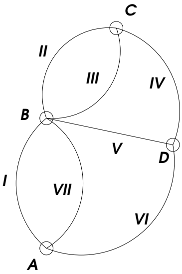
Blocks World



Königsberg Bridge Problem

Is it possible to make a tour so that one passes just once over all the bridges over the river Preger in Königsberg ?

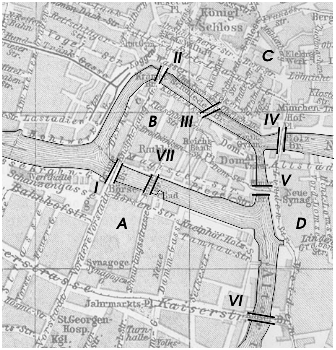


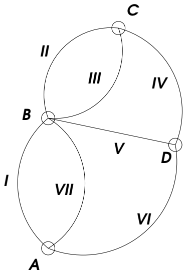


An Eulerian circuit is a graph cycle which uses each graph edge exactly once. A connected graph has an Eulerian circuit iff it has no graph vertices of odd degree.

Königsberg Bridge Problem

Is it possible to make a tour so that one passes just once over all the bridges over the river Preger in Königsberg ?

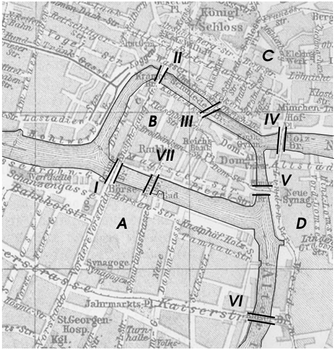


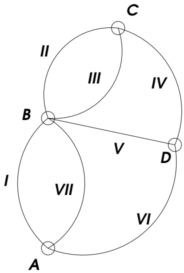


An Eulerian circuit is a graph cycle which uses each graph edge exactly once. A connected graph has an Eulerian circuit iff it has no graph vertices of odd degree.

Königsberg Bridge Problem

Is it possible to make a tour so that one passes just once over all the bridges over the river Preger in Königsberg ?

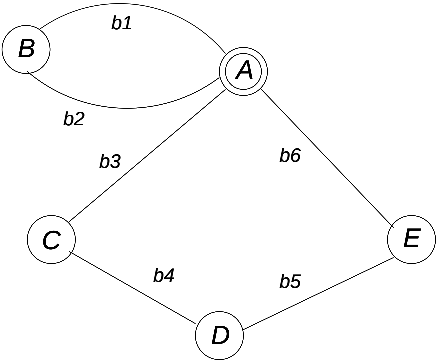




An Eulerian circuit is a graph cycle which uses each graph edge exactly once. A connected graph has an Eulerian circuit iff it has no graph vertices of odd degree.

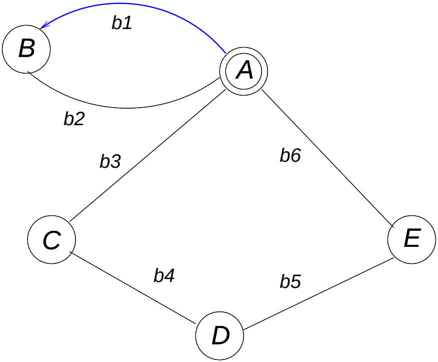
Eulerian circuit

Exitence proof not satisfactory. Need to built up a solution = a plan.



Eulerian circuit

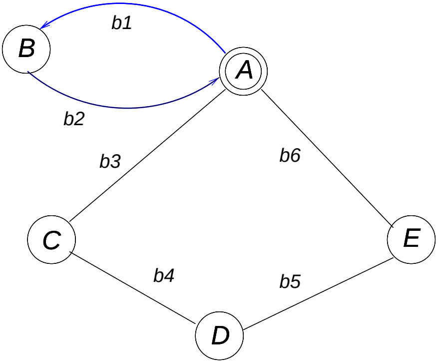
Exitence proof not satisfactory. Need to built up a solution = a plan.



CROSS_b1_A_B

Eulerian circuit

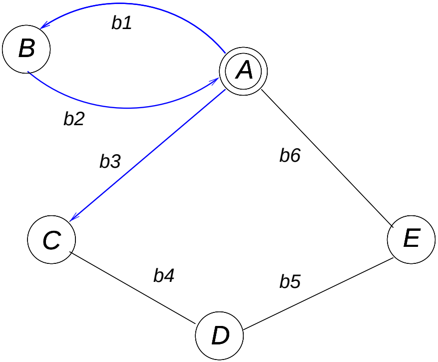
Exitence proof not satisfactory. Need to built up a solution = a plan.



CROSS_b1_A_B
CROSS_b2_B_A

Eulerian circuit

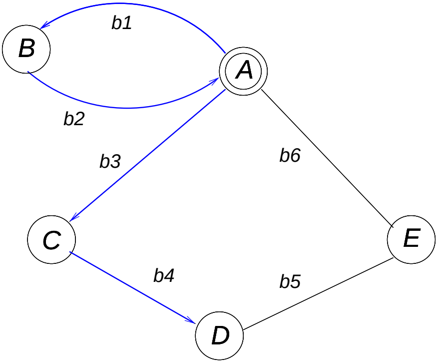
Exitence proof not satisfactory. Need to built up a solution = a plan.



CROSS_b1_A_B
CROSS_b2_B_A
CROSS_b3_A_C

Eulerian circuit

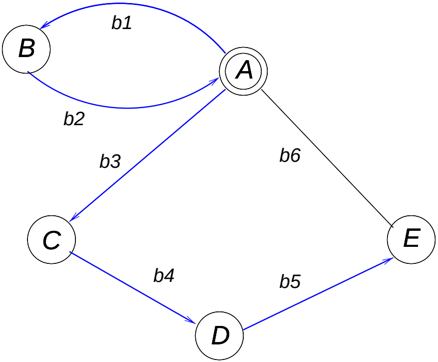
Exitence proof not satisfactory. Need to built up a solution = a plan.



CROSS_b1_A_B
CROSS_b2_B_A
CROSS_b3_A_C
CROSS_b4_C_D

Eulerian circuit

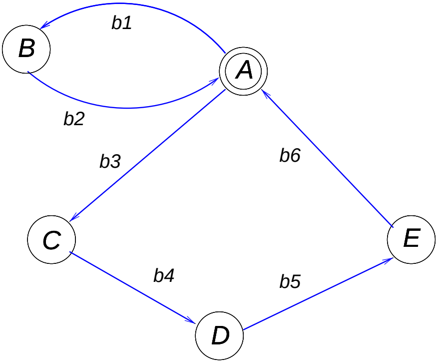
Exitence proof not satisfactory. Need to built up a solution = a plan.



CROSS_b1_A_B
CROSS_b2_B_A
CROSS_b3_A_C
CROSS_b4_C_D
CROSS_b5_D_E

Eulerian circuit

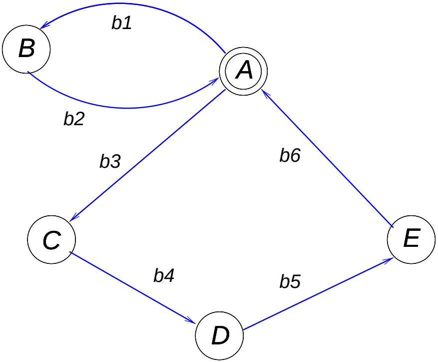
Exitence proof not satisfactory. Need to built up a solution = a plan.



CROSS_b1_A_B
CROSS_b2_B_A
CROSS_b3_A_C
CROSS_b4_C_D
CROSS_b5_D_E
CROSS_b6_E_A

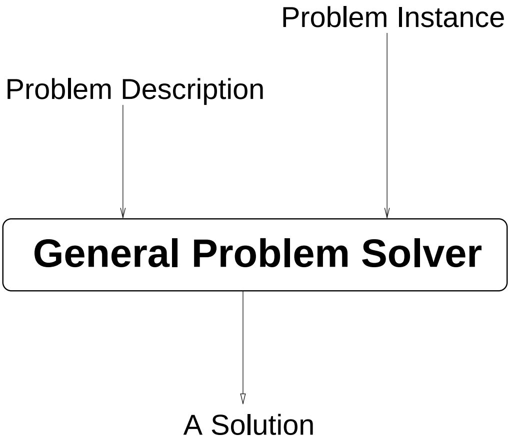
Eulerian circuit

Exitence proof not satisfactory. Need to built up a solution = a plan.

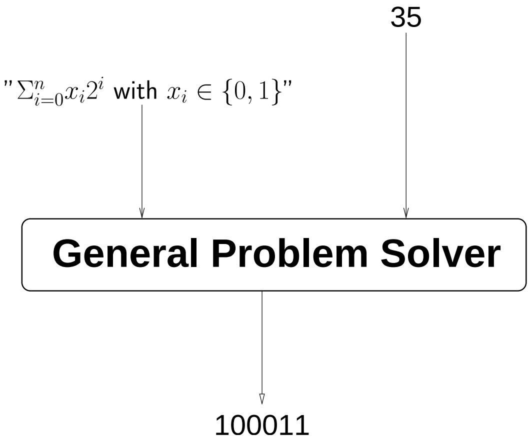


CROSS_b1_A_B
CROSS_b2_B_A
CROSS_b3_A_C
CROSS_b4_C_D
CROSS_b5_D_E
CROSS_b6_E_A

General Problem Solver



General Problem Solver



General Problem Solver

$(x_1 \vee x_3 \vee \bar{x}_4) \wedge (x_4) \wedge (x_2 \vee \bar{x}_3)$

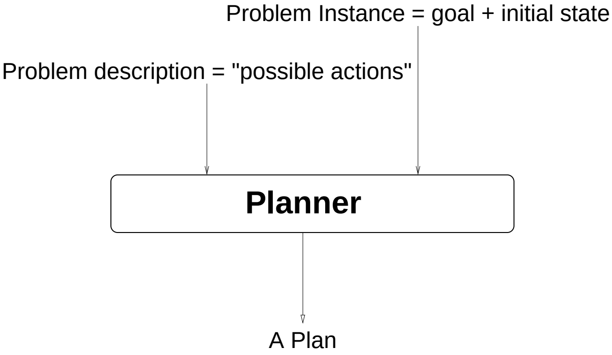
"$\models \bigwedge C_i$ with $C_i = \bigvee l_j$
where l_j is either a positive or a negative literal"

General Problem Solver

$(x_1 \wedge x_4 \wedge x_2)$

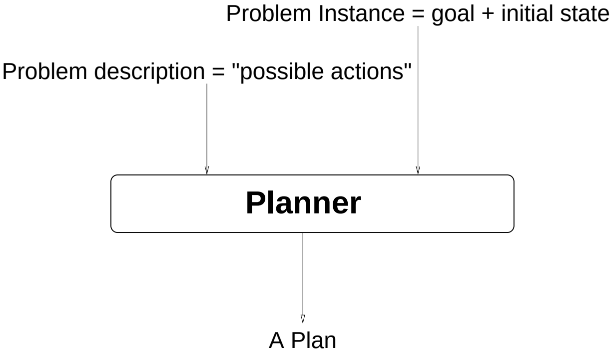
Mean-Ends Reasoning

- Planning is mean-ends reasoning : the process of deciding how to achieve a goal with available actions
- A planning algorithm outputs a plan = an ordered sequence of actions that drives the **agent** from the initial state of the world to the targeted goal



Mean-Ends Reasoning

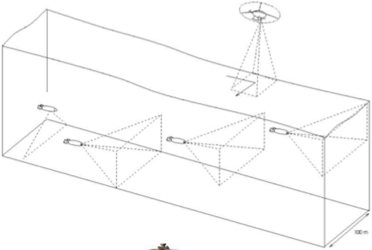
- Planning is mean-ends reasoning : the process of deciding how to achieve a goal with available actions
- A planning algorithm outputs a plan = an ordered sequence of actions that drives the **agent** from the initial state of the world to the targeted goal

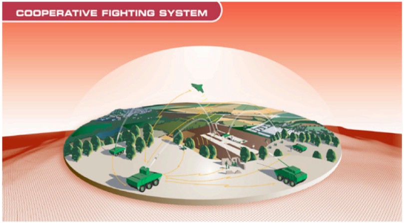


Autonomous robots

Drones

Underwater Autonomous Vehicles

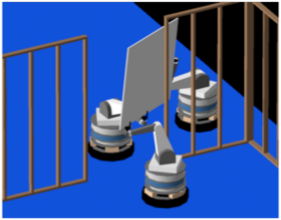


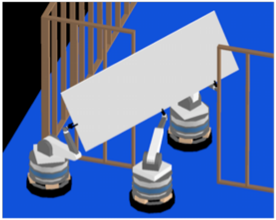


Satellite constellation

Robotics experiments

Coordinated control of multiple robot manipulators

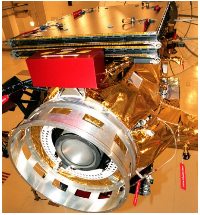




RoboCup

Deep Space 1's Remote Agent

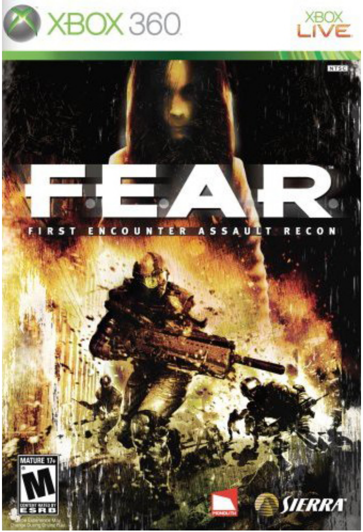
"Remote Agent (remote intelligent self-repair software)(RAX), developed at NASA Ames Research Center and JPL, was the first artificial intelligence control system to control a spacecraft without human supervision. Remote Agent successfully demonstrated the ability to plan onboard activities and correctly diagnose and respond to simulated faults in spacecraft components. Autonomous control will enable future spacecraft to operate at greater distances from Earth, and to carry out more sophisticated science-gathering activities in deep space.



Major components of Remote Agent were a robust planner (EUROPA), a plan execution system (EXEC) and a model-based diagnostic system (Livingstone). EUROPA was used as a ground-based planner for the Mars Exploration Rovers. EUROPA II was used to support the Phoenix Mars Lander and will support the upcoming Mars Science Laboratory."

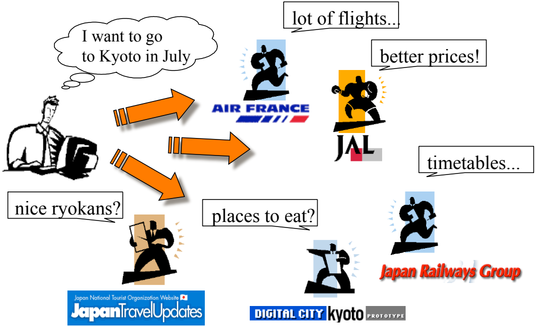
"Deep Space 1", Wikipedia.

Some video games



Cooperating web services

This is a composition of travel services. The user has a goal : "I want to go to Kyoto in July" that is sent to a set of travel services like Air France, Japan Air Line, hotel online booking, touristic information etc. These services compose their skills to solve the goal and return a solution plan that says...



Cooperating web services

This is a composition of travel services. The user has a goal : "I want to go to Kyoto in July" that is sent to a set of travel services like Air France, Japan Air Line, hotel online booking, touristic information etc. These services compose their skills to solve the goal and return a solution plan that says...

Cooperating web services

This is a composition of travel services. The user has a goal : "I want to go to Kyoto in July" that is sent to a set of travel services like Air France, Japan Air Line, hotel online booking, touristic information etc. These services compose their skills to solve the goal and return a solution plan that says...

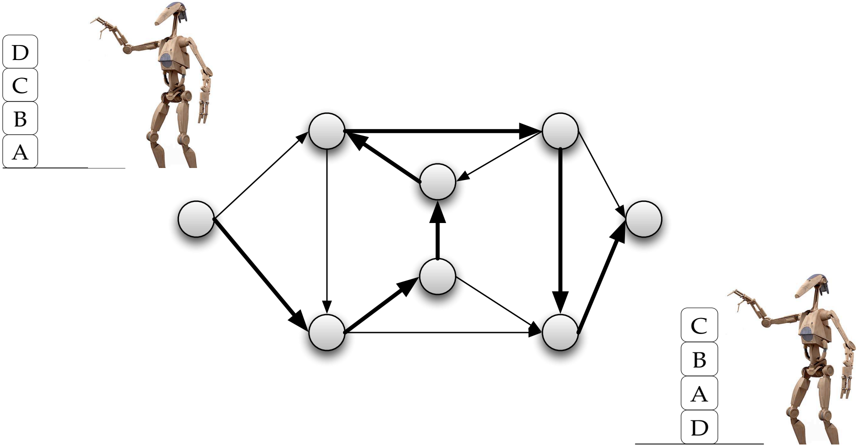
State Transition System

The conceptual model of planning can be represented as a state transition system. Formally, a 3-tuple $\Sigma = (S, A, \gamma)$ where :

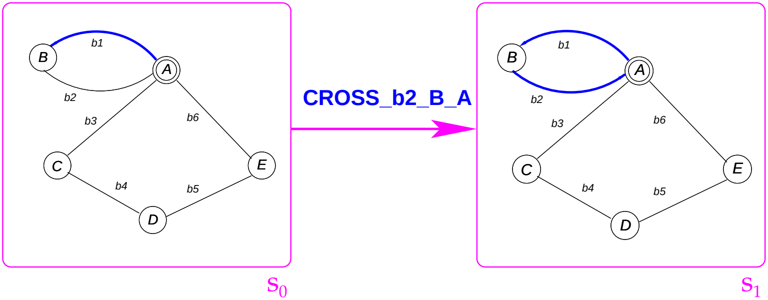
- $S = s_1, s_2, \cdots, s_n$ is a finite or recursively enumerable set of states
- $A = a_1, a_2, \cdots, a_n$ is a finite or recursively enumerable set of actions
- $\gamma : S \times A \rightarrow S$ is a state transition function

A state transition system can be represented by a directed graph whose nodes are the state in S. *Situations of the world* are represented with states.

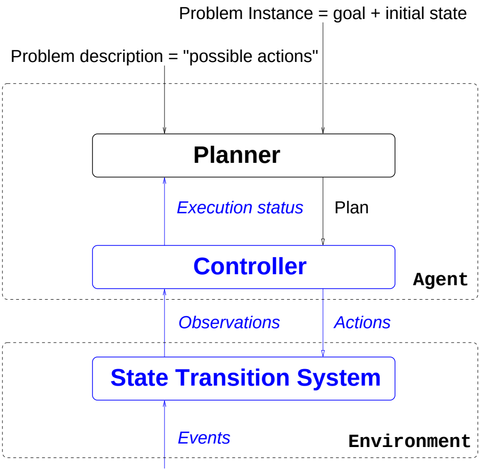
State Transition System



State Transition System

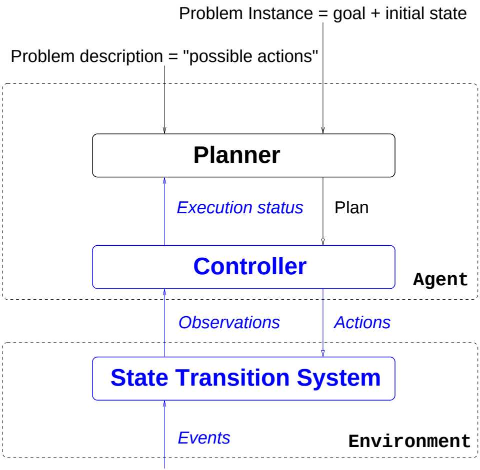


Autonomous Agent



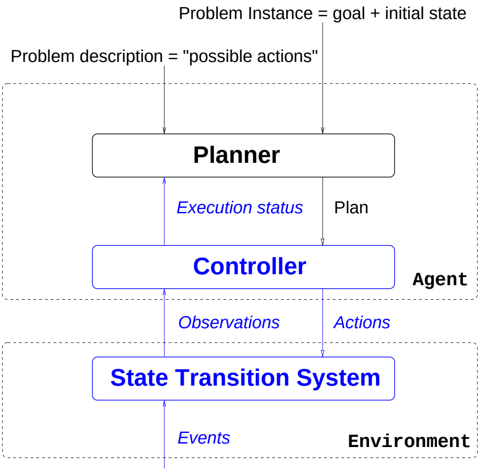
- State transition system Σ evolves as specified by its state transition function according to the events and actions that it receives. This system represents the agent's environment
- Agent = artefact composed of a controller and a planner
- Controller = given as input the state s of the system, provides as output an action a according to some plan
- Planner = given as input a problem description, an initial situation and some objective, synthesizes a plan for the controller in order to achieve the objectives

Autonomous Agent



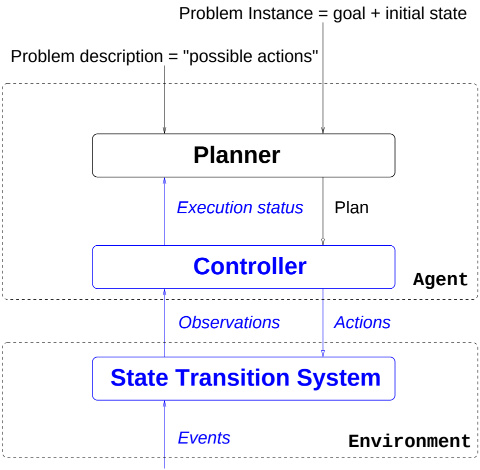
- State transition system Σ evolves as specified by its state transition function according to the events and actions that it receives. This system represents the agent's environment
- Agent = artefact composed of a controller and a planner
- Controller = given as input the state s of the system, provides as output an action a according to some plan
- Planner = given as input a problem description, an initial situation and some objective, synthesizes a plan for the controller in order to achieve the objectives

Autonomous Agent



- State transition system Σ evolves as specified by its state transition function according to the events and actions that it receives. This system represents the agent's environment
- Agent = artefact composed of a controller and a planner
- Controller = given as input the state s of the system, provides as output an action a according to some plan
- Planner = given as input a problem description, an initial situation and some objective, synthesizes a plan for the controller in order to achieve the objectives

Autonomous Agent



- State transition system Σ evolves as specified by its state transition function according to the events and actions that it receives. This system represents the agent's environment
- Agent = artefact composed of a controller and a planner
- Controller = given as input the state s of the system, provides as output an action a according to some plan
- Planner = given as input a problem description, an initial situation and some objective, synthesizes a plan for the controller in order to achieve the objectives

Some restrictions

- Considered problems = State Transition Systems
- Environments = State Transition Systems :
  - Finite, Σ has a finite set of states
  - Fully observable, one has complete knowledge about the state of Σ
  - Deterministic, for every states s and for every event of applicable action a, its application brings to a single other state.
  - Static, Σ has no internal dynamics

Some restrictions

- Considered problems = State Transition Systems
- Environments = State Transition Systems :
  - Finite, Σ has a finite set of states
  - Fully observable, one has complete knowledge about the state of Σ
  - Deterministic, for every states s and for every event of applicable action a, its application brings to a single other state.
  - Static, Σ has no internal dynamics

Some restrictions

- Goals = a world state
- Sequential Plans : a solution plan to a planning problem is a linearly ordered finite sequence of actions
- Implicit time : actions and events have no duration (= state transition systems do not represent time explicitly)
- Offline Planning, not concerned with any change that may occur in Σ. Plans given for initial and goal states regardless of the ongoing dynamics, if any.

Some restrictions

- Goals = a world state
- Sequential Plans : a solution plan to a planning problem is a linearly ordered finite sequence of actions
- Implicit time : actions and events have no duration (= state transition systems do not represent time explicitly)
- Offline Planning, not concerned with any change that may occur in Σ. Plans given for initial and goal states regardless of the ongoing dynamics, if any.

Some restrictions

- Goals = a world state
- Sequential Plans : a solution plan to a planning problem is a linearly ordered finite sequence of actions
- Implicit time : actions and events have no duration (= state transition systems do not represent time explicitly)
- Offline Planning, not concerned with any change that may occur in Σ. Plans given for initial and goal states regardless of the ongoing dynamics, if any.

Some restrictions

- Goals = a world state
- Sequential Plans : a solution plan to a planning problem is a linearly ordered finite sequence of actions
- Implicit time : actions and events have no duration (= state transition systems do not represent time explicitly)
- Offline Planning, not concerned with any change that may occur in Σ. Plans given for initial and goal states regardless of the ongoing dynamics, if any.

State Definition

Definition (State)

Let $L = \{p_1, \cdots, p_n\}$ be a finite set of proposition symbols. A state s is a subset of L. If $p \in s$ then p holds in s. Otherwise p does not hold in s = *Closed World Assumption*

Problem Definition

Example (States, STRIPS notations [Fikes and Nilsson, 1971])

(preconds
(at A token)
(connected b1 A B)
(connected b1 B A)
(connected b2 A B)
(connected b2 B A)
(connected b3 A C)
(connected b3 C A)
(connected b4 C D)
(connected b4 D C)
(connected b5 D E)
(connected b5 E D)
(connected b6 E A)
(connected b6 A E))

(effect
(at A token)
(clear b1)
(clear b2)
(clear b3)
(clear b4)
(clear b5)
(clear b6))

Domain Definition

Definition (Domain)

A planning domain on L is a restricted state transition system $\Sigma = (S, A, \gamma)$ such that :

- $S \subseteq 2^L$, i.e., each state s is a subset of L
- Each action $a \in A$ is such that $a = (precond(a), effect^-(a), effect^+(a))$ and $effect^-(a) \cap effect^+(a) = \emptyset$
- $a \in A$ is applicable to $s \in S$ iff $precond(a) \subseteq s$

Problem Definition

Example (Actions, STRIPS notations [Fikes and Nilsson, 1971])

(operator CROSS_b2_B_A
(params)
(preconds
(at B token)
(connected b2 B A))
(effects
(at A token)
(clear b2)
(del at B token)
(del connected b2 A B)
(del connected b2 B A)))

$\vdots$

Domain Definition

- Preconditions = things that must be true about the world before the action is performed
- Effects (post-conditions) = things that the corresponding action guarantees will be true about the world after it has been performed

$$s_{i+1} = \gamma(s_i, a) = (s_i - effect^-(a)) \cup effect^+(a)$$

if a is applicable to s

- If $s_i \in S$ then $\gamma(s_i, a) \in S$

Problem Definition

Definition (Problem)

A planning problem is a triple $\mathcal{P} = (\Sigma, s_0, g)$ where $s_0 \in S$ is the initial state, $g \subseteq L$ is a set of propositions representing the goals to achieve. The set of goal states is $S_g = \{s \in S | s \text{ satisfies } g\}$. *$s$ satisfies g* ($s \models g$) iff $g \subseteq s$.

Plan Definition

Definition (Plan)

A plan is any sequence of action $\pi = [a_1, \cdots, a_k]$ $(k \geq 0)$. $|\pi| = k$ is the length of the plan. If $\pi_1 = [a_1, \cdots, a_k]$ and $\pi_2 = [a'_1, \cdots, a'_j]$ are plans, then their concatenation is the plan $\pi_1 \cdot \pi_2 = [a_1, \cdots, a_k, a'_1, \cdots, a'_j]$.

Plan Definition

The state produced by applying π to a state s is the state produced by applying the actions of π sequentially. We will denote this by extending the state transition function as follows :

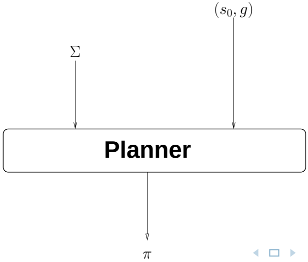
$$\gamma(s,\pi) = \begin{cases} s, & \text{if } k = 0 \\ \gamma(\gamma(s,a_1),[a_2,\cdots,a_k]), & \text{if } k > 0 \text{ and } a_1 \text{ is applicable to } s \\ \perp, & \text{undefined otherwise} \end{cases}$$

Solution Plan Definition

Definition (Solution Plan)

Let $\mathcal{P} = (\Sigma, s_0, g)$ be a planning problem. A plan π is a solution for $\mathcal{P}$ iff $g \subseteq \gamma(s_0, \pi)$.

- A solution plan π is *redundant* if a proper subsequence of π is also a solution of $\mathcal{P}$
- A solution plan π is *minimal* if no other solution plan for $\mathcal{P}$ contains fewer actions that π



Solution Plan Definition

Example (Solution Plans)

- $\pi_1 =$ [CROSS_b1_A_B, CROSS_b2_B_A] is not a solution plan
- $\pi_2 =$ [CROSS_b1_A_B, CROSS_b2_B_A, CROSS_b3_A_C, $\cdots$, CROSS_b6_E_A] is a solution plan

Representation Extension

- Propositions have a content, which is a tuple of *elements*
- Elements = *variables* or *constants* ; there are infinitely many of each
- Functions, propositional operators and quatification are not allowed
- Propositions can be *negated*. Two propositions are *negations* iff one is negated and the other one is not + have the same content : `(on a ?x)` and `¬(on a ?x)`
- *Codesignation* = equivalence relation on variables and constants. In a plan, each variable must be constrained to codesignate with a constant. *Binding constraints* enforce codesignation or noncodesignation of elements.
- Distinct constants may not codesignate. Two propositions codesignate if both are negated or both are not, if their contents are of the same length, and if corresponding elements codesignate : `(on a ?x)` and `(on a ?y)` codesignate iff $?x$ and $?y$ codesignate.

Planning Operators and Actions

- A planning operator is a triple $o = (name(o), precond(o), effects^{+}(o), effects^{-}(o))$: the name of the operator is a syntactic expression of the form $n(x_1, \cdots, x_k)$. $precond(o)$, $effects^{+}(o)$ and $effects^{-}(o)$ are generalizations of the preconditions and the effects i.e. sets of atoms (propositions possibly negated with contents)
- Actions are ground instantiations of operators = each variable codesignate with a constant
- γ is unchanged but it is illegal for a proposition to be both asserted and denied in s
- A planning problem is a tuple $\mathcal{P} = (\mathcal{O}, s_0, g)$ where $\mathcal{O}$ is the set of operators

Planning Operators and Actions

Example (CROSS Operator)

(b1 EDGE)
(b2 EDGE)
(b3 EDGE)
(b4 EDGE)
(b5 EDGE)
(b6 EDGE)
(A VERTEX)
(B VERTEX)
(C VERTEX)
(D VERTEX)
(E VERTEX)
⋮

```
(operator CROSS
   (params
      (<from> VERTEX)
      (<to> VERTEX)
      (<edge> EDGE))
   (preconds
      (at <from> token)
      (connected <edge> <from> <to>))
   (effects
      (at <to> token)
      (clear <edge>)
      (del at <from> token)
      (del connected <edge> <to> <from>)
      (del connected <edge> <from> <to>)))
```

Planning Operators and Actions

Example

```
(operator CROSS
    (params
        (A VERTEX)
        (B VERTEX)
        (b1 EDGE))
    (preconds
        (at A token)
        (connected b1 A B))
    (effects
        (at B token)
        (clear b1)
        (del at A token)
        (del connected b1 B A)
        (del connected b1 A B)))
```

```
(operator CROSS
    (params
        (A VERTEX)
        (B VERTEX)
        (b2 EDGE))
    (preconds
        (at A token)
        (connected b2 A B))
    (effects
        (at B token)
        (clear b2)
        (del at A token)
        (del connected b2 B A)
        (del connected b2 A B)))
```

Blocks World (STRIPS Notation)

(blockA OBJECT)
(blockB OBJECT)
(blockC OBJECT)
(blockD OBJECT)
(preconds
  (on-table blockA)
  (on blockB blockA)
  (on blockC blockB)
  (on blockD blockC)
  (clear blockD)
  (arm-empty))
(effects
  (on blockB blockA)
  (on blockC blockB)
  (on blockA blockD))

Blocks World (STRIPS Notation)

```
(operator PICK-UP
    (params (<ob> OBJECT))
    (preconds
        (clear <ob>)(on-table <ob>)(arm-empty))
    (effects
        (holding <ob>)
        (del clear <ob>)(del on-table <ob>)(del arm-empty)))
```

Blocks World (STRIPS Notation)

(operator PUT-DOWN
(params (<ob> OBJECT))
(preconds
(holding <ob>))
(effects
(clear <ob>)(arm-empty)(on-table <ob>)
(del holding <ob>)))

Blocks World (STRIPS Notation)

```
(operator STACK
    (params (<ob> OBJECT)(<underob> OBJECT))
    (preconds
        (clear <underob>)(holding <ob>))
    (effects
        (arm-empty)(clear <ob>)(on <ob> <underob>)
        (del clear <underob>)(del holding <ob>)))
```

Blocks World (STRIPS Notation)

```
(operator UNSTACK
    (params (<ob> OBJECT)(<underob> OBJECT))
    (preconds
        (on <ob> <underob>)(clear <ob>)(arm-empty))
    (effects
        (holding <ob>)(clear <underob>)
        (del on <ob> <underob>)(del clear <ob>)
        (del arm-empty)))
```

Blocks World

1 UNSTACK_blockD_blockC
2 PUT-DOWN_blockD
3 UNSTACK_blockC_blockB
4 PUT-DOWN_blockC
5 UNSTACK_blockB_blockA
6 STACK_blockB_blockC
7 PICK-UP_blockA
8 STACK_blockA_blockD
9 UNSTACK_blockB_blockC
10 STACK_blockB_blockA
11 PICK-UP_blockC
12 STACK_blockC_blockB
3 entries in hash table, 3 hash hits, avg set size 5.
17 total set-creation steps (entries + hits + plan length - 1).
13 actions tried
0.08 secs

For Further Readings

A. Newell. and H. A. Simon.
GPS : A Program that Simulates Human Thought.
in Feigenbaum and Feldman (eds), "Computers and Thought", McGraw-Hill, New York, 1963, pp. 279–293.

R. E. Fikes and N. J. Nilsson.
STRIPS : a new approach to the application of theorem proving to problem solving.
"Artificial Intelligence", 2(3–4) :189–208, 1971.

M. Ghallab, D. Nau and P. Traverso.
Automated Planning, theory and pratice.
Morgan Kaufmann, 2004.