Advanced Techniques in Artificial Intelligence

Curso 2021-2022

German Rigau

german.rigau@ehu.eus

Grado en Ingeniería en Informática

Topics

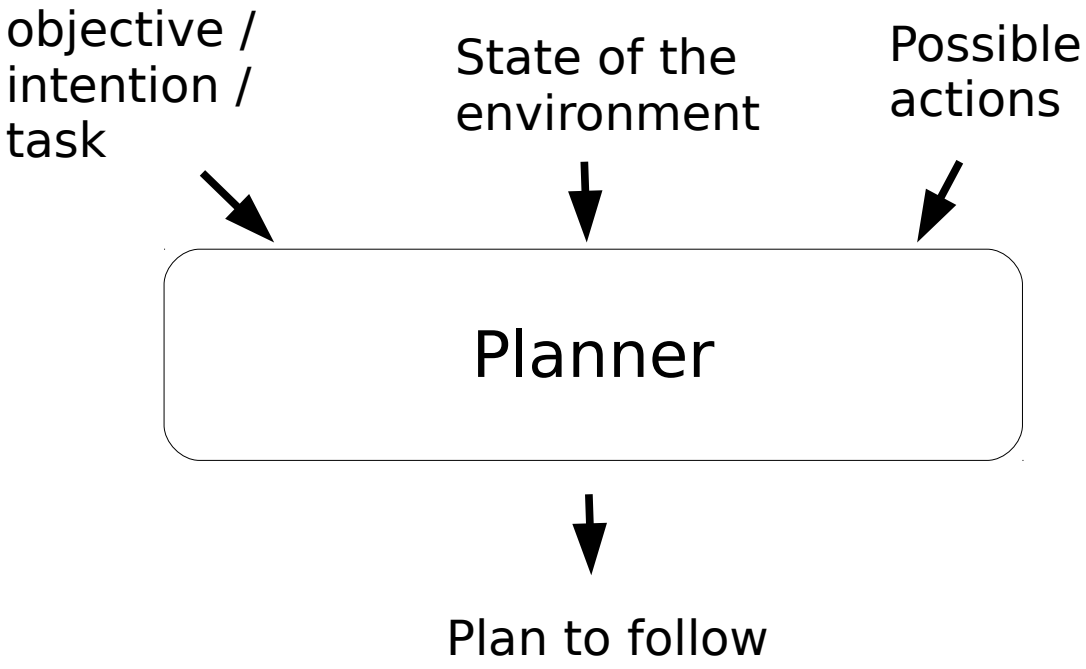
- Intelligent Agents
- Multiagent Systems
- Planning

3 Planning

1. Introduction
2. Classical planning
3. Real world planning

Introduction

- Since the early '70s, the AI community specialized in planning has been concerned with the design problem of artificial agents capable of acting in an environment.
- Planning can be seen as a form of automatic programming: the design of a course of action that will satisfy a certain objective.
- Within the symbolic AI community, it has been assumed for some time that some type of planning system should be part of the core components of any artificial agent
- The basic idea is to provide the planning agent:
  - representation of the objective to be achieved
  - representation of the actions you can perform
  - representation of the environment
  - ability to generate a plan to achieve the objective



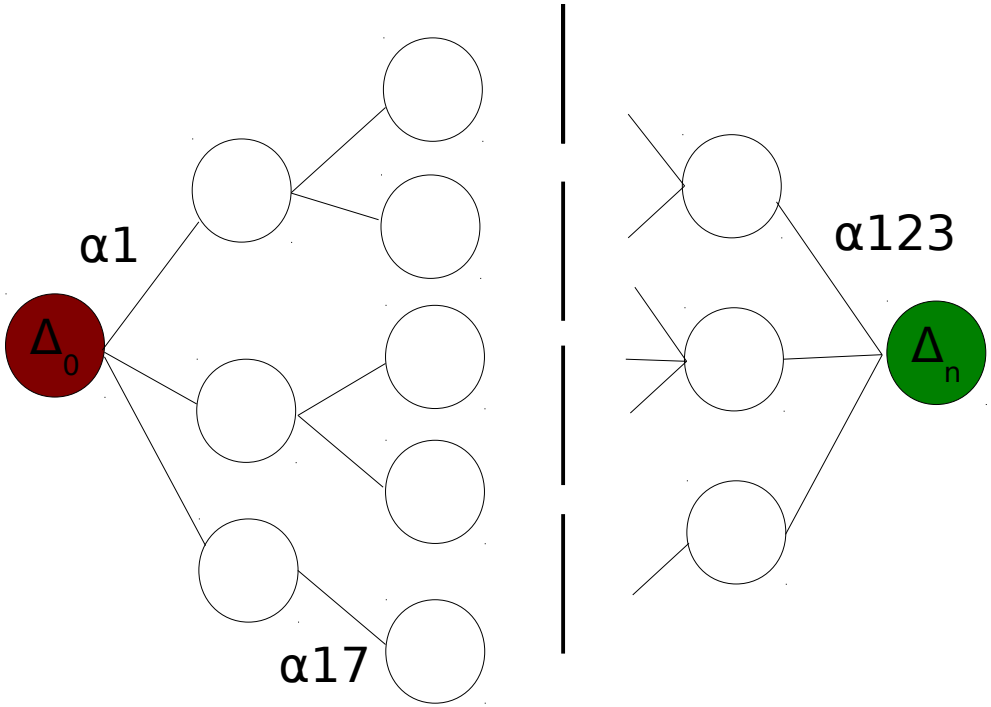
How to represent …

- Objective to achieve
- State of the environment
- Actions available for the agent
- The plan itself

KRR: Knowledge Representation and Reasoning

Introduction

- A plan is a sequence (list) of actions, from an initial state to a final state.
- Planning can be seen as a search problem in a state space.



Introduction

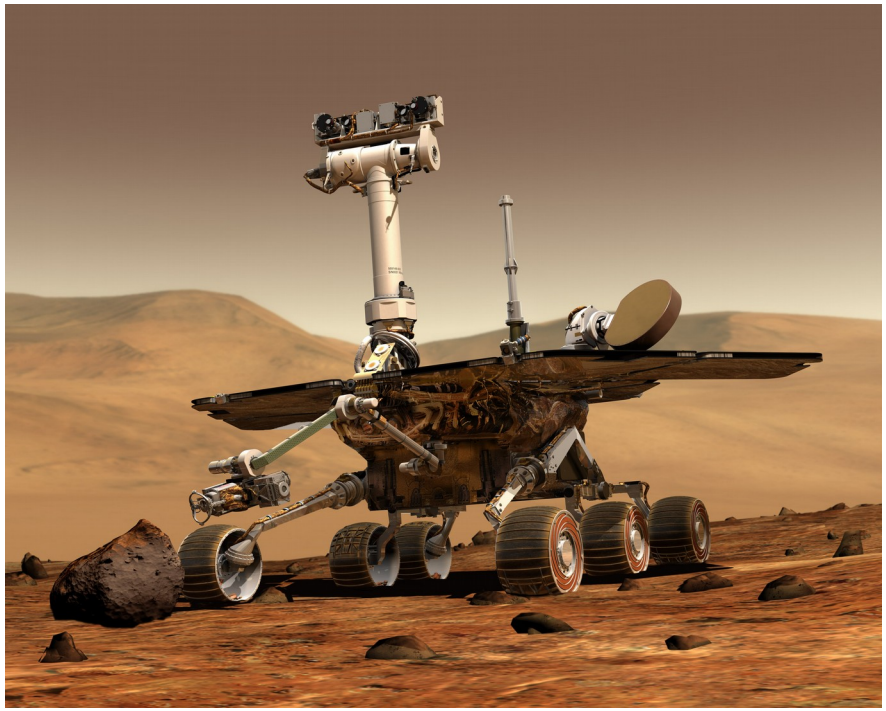
- Classic search algorithms are interested only in returning the final state or solution-state.
- Planning algorithms are not only interested in finding the solution state, but in maintaining all the intermediate states that lead from the initial state to the end.
- Planning algorithms often use not only the knowledge within the heuristic, but also the descriptions of the effects of the actions to guide their search (they use the logical structure of the problem).
- Many planning algorithms reduce the complexity of the problem by decomposing it into sub-objectives
  - This can only be done in real problems that are decomposable or quasi-decomposable (the planner breaks down the problem and then resolves small conflicts when recomposing the solution)

Introduction

- Our vision of the world is incomplete: limited rationality
- The world changes constantly: dynamism
- The world is not deterministic: uncertainty
- The actions take time to execute: temporary reasoning
- Our goals are contradictory: dependency between goals
- Not all plans are good: quality
- The plans are not always valid: execution and replanning
- Adaptation to the world: learning
- Planning and philosophy: beliefs, intentions and desires

Introduction

- Mars Exploration Rovers [NASA]
  - Planning of the tasks to be performed during a Martian day is automatically carried out by a program based on the exploration objectives set by the mission personnel on Earth.



Introduction

- Types of Planners:
  - Domain-specific planners:
    - Specifically designed for a domain and can hardly be used in other domains: many practical applications.
  - Domain independent planners:
    - The planning mechanism is general enough to be used in domains that meet certain restrictions: Not efficient and Not practical applications
  - Configurable planners to the domain:
    - The planning mechanism is domain independent, but the entry to the scheduler includes knowledge of the domain to restrict the search of the scheduler.

Classical planning

- Restrictions of the domain (1):
  - Restriction 0: **finite**. Finite set of states.
  - Restriction 1: **fully observable**. You have a complete knowledge of the environment. The planner perfectly perceives the state of the environment and the effect of its actions on the environment
  - Restriction 2: **deterministic**. You can predict and predefine the effects of all actions).
  - Restriction 3: **static**. The changes happen only when the planning agent acts.
  - Restriction 4: **discrete**. The environment can be described discreetly:
    - Time, actions, objects, effects, ...

Classical planning

- Restrictions of the domain (2):
  - Restriction 5: **implicit time**. The actions have no duration, the transition states are instantaneous. They do not represent time explicitly.
  - Restriction 6: **sequential plans**. A solution is a linearly ordered sequence of actions.
  - Restriction 7: **Offline planning**. The planner does not take into account any changes that may occur in the environment while it is planning. Plan according to the initial state and given objectives without observing the changes, if any.
  - Restriction 8: **achievable goal** (!)

Classical planning

- $Ac = \{\alpha_1, \ldots, \alpha_n\}$: set of actions
- $< P_\alpha, D_\alpha, A_\alpha >$ is descriptor of one action $\alpha \in Ac$
- P_α is a set of formulas that characterize the precondition of action α
- D_α is a set of formulas that characterize those facts that become false by the execution of α ('delete list')
- A_α is a set of formulas that characterize those facts that become true by the execution of α (‘add list’)
- $\pi = (\alpha_1, \ldots, \alpha_n)$ is a plan

Classical planning

- Open representation of knowledge about states, objectives and actions.
  - Formal language
    - Ex: predicate logic, First Order Logic, ...
  - States and goals (objectives) are represented by sets of logical statements
    - Ex: *in(p1, bcn), plane(p1), ...*
    - It is common in some planners that what does not appear explicitly represented in a state is false: closed world assumption.
  - The actions are represented by logical descriptions of preconditions and effects
    - Ex: action*(fly(P, O, T),*

      PRECOND: in(P,O)∧plane(P)∧airport(O)∧airport(T),

      EFFECT: ¬in(P,O)∧in(P,T))

Classical planning

- STRIPS (STanford Research Institute Problem Solver) (Fikes & Nilsson 1971) was one of the first of the planning systems.
- For certain real problems it has been shown that STRIPS does not have sufficient expressivity.
- ADL (Action Description Language) expands STRIPS ideas
- Notations of STRIPS and ADL are adecuate for most real problems and domains.

Classical planning

- Since 1998, the planning research community has developed a standard language for describing plans: Planning Domain Description Language (PDDL)
- Initial objective: common language for global competition of planners.
- Nowadays it has become a de facto standard
- WARNINGS:
  - there are several versions of PDDL, from 1.0 to 3.1, each with different levels of expressiveness
  - There is no planner that supports the complete specification 3.1, but subsets of it.

Classical planning: STRIPS

- Representation of states: planners decompose the world into logical conditions, representing a state as a conjunction of positive literals:
  - Propositions: *poor* ∧ *boring*
  - FOL literals: *in(plain1, bcn)* ∧ *in(plain2, jfk)*
- Representation of objectives: an objective is a partially specified state
  - An state s satisfy an objective o if s contains all atoms from o (and possibly some other)
    - Ex: state (*rich* ∧ *famous* ∧ *handsome)* satisfies objective (*rich* ∧ *famous)*

Classical planning: STRIPS

- Representation of actions: the actions are specified in terms of the preconditions that must be fulfilled before they can be executed and the effects they produce once they have been executed
  - Ex: action*(fly(P, O, T),*
    PRECOND: in(P,O)∧plane(P)∧airport(O)∧airport(T),
    EFFECT: ¬in(P,O)∧in(P,T))
- The precondition is a conjunction of positive literals that specifies that it must be true in a state before the action is executed. All the variables in the precondition must appear in the list of parameters of the action.
- The effect is a conjunction of literals describing how the current state changes when the action is executed. All variables must also appear in the action parameter list.

Classical planning: STRIPS

- An action is applicable to any state that satisfies the precondition
  - In FOL: there is a substitution for the variables in the precondition. For example, the state:

    in(p1, jfk) ∧ plane(p1) ∧ in(p2, bcn) ∧ plane(p2) ∧ airport(jfk) ∧ airport(bcn)
  - satisfies the precondition of the flying action:

    in(P, O) ∧ plane(P) ∧ airport(O) ∧ airport(D)
- The result of executing the action in a state s is a state s' to which the positive literals of the effect are added and the negative literals are eliminated
  - For example, the effect of the fly action on the previous state:
    - *in(p1, jfk) ∧ plane(p1) ∧ in(p2, bcn) ∧ plane(p2) ∧ airport(jfk) ∧ airport(bcn)*
  - Removed: *in(p1,jfk)*

Example STRIPS: Air cargo transportation

- Two cargo (c1 and c2) are in 2 airports (bcn, jfk)
- We have two planes (p1 and p2) to transport the loads, one in each airport
- We describe the initial state like this:
  - *start(in(c1, bcn) ∧ in(c2, jfk) ∧ in(p1, bcn) ∧ in(p2, jfk) ∧ cargo(c1) ∧ cargo(c2) ∧ plane(p1) ∧ plane(p2) ∧ airport(bcn) ∧ airport(jfk))*
- The goal at the end is have c1 in jfk and c2 in bcn
- We describe the objective as follows:
  - *goal(in(c1, jfk) ∧ in(c2, bcn))*

Example STRIPS: Air cargo transportation

- We describe the actions of loading, downloading and flying:
  - *action(load(C, P, A),*

    PRECOND: in(C, A) ∧ in(P, A) ∧ cargo(C) ∧ plane(P) ∧ airport(A)

    EFFECT: ¬in(C, A) ∧ inside(C, P))
  - *action(download(C, P, A),*

    PRECOND: inside(C, P) ∧ in(P, A) ∧ cargo(C) ∧ plane(P) ∧ airport(A)

    EFFECT: in(c, AE) ∧ ¬inside(C, A))
  - *action(fly(P, O, T),*

    PRECOND: in(P, O) ∧ plane(P) ∧ airport(O) ∧ airport(T)

    EFFECT: ¬in(P, O) ∧ in(P, T))

Example STRIPS: Air cargo transportation

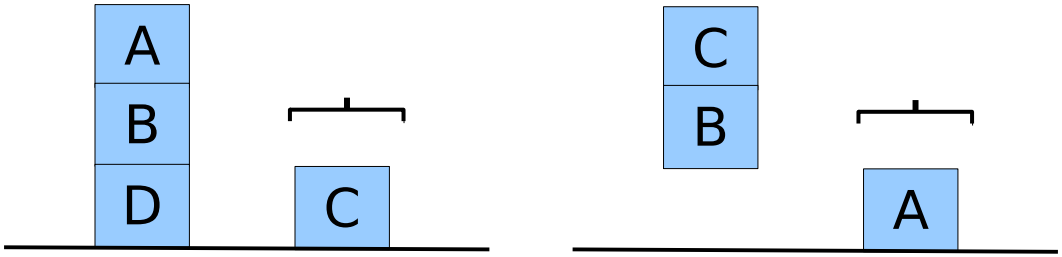
- Solution: the plan is composed of a sequence of actions.
- In this case there are several solutions:
  - Ex. Solution 1: we use the two planes to make the transfer

    [load(c1, p1, bcn), fly(p1, bcn, jfk), download(c1, p1, jfk), load(c2, p2, jfk), fly(p2, jfk, bcn), download(c2, p2, bcn)]
  - Ex. Solution 2: we use just one plane

    [load(c1, p1, bcn), fly(p1, bcn, jfk), download(c1, p1, jfk), load(c2, p1, jfk), fly(p1, jfk, bcn), download(c2, p1, bcn)]

STRIPS example: block world

- A set of blocks, a table and a robot arm.
- All blocks are equal in size, shape and color. They differ in name.
- The table has unlimited extension
- Each block can be on top of the table, on top of a single block or held by the robot's arm.
- The robot's hand can only hold one block at a time.
- Troubleshooting involves moving from an initial configuration (state) to a state where certain goals are achieved.



Initial state

Final state

STRIPS example: block world

- The following predicates could be used:
  - on(X, Y) : block X is over block Y.
  - in_table(X) : block X is on the table.
  - free(X) : block X has no block over.
  - in_hand(X) : the robot holds the block X
  - free_hand : the robot's hand is free (not holding any block)
- Initial state
  - on(a, b), on(b, d), in_table(d), in_table(c), free(a), free(c), free_hand
- Goals:
  - in_table(a), on(c, b)

STRIPS example: block world

- UNSTACK(X, Y)
  precondition: on(X,Y), free(X), free_hand
  added: in_hand(X), free(Y)
  removed: on(X, Y), free(X), free_hand
- TAKE(X)
  precondition: in_table(X), free(X), free_hand
  added: in_hand(X)
  removed: in_table(X), free(X) , free_hand
- STACK(X, Y)
  precondition: in_hand(X), free(Y)
  added: on(X, Y), free(X), free_hand
  removed: in_hand(X), free(Y)
- LEAVE(X)
  precondition: in_hand(X)
  added: in_table(X), free(X), free_hand
  removed: in_hand(X)

Example: The Dock Worker Robots (DWR) domain

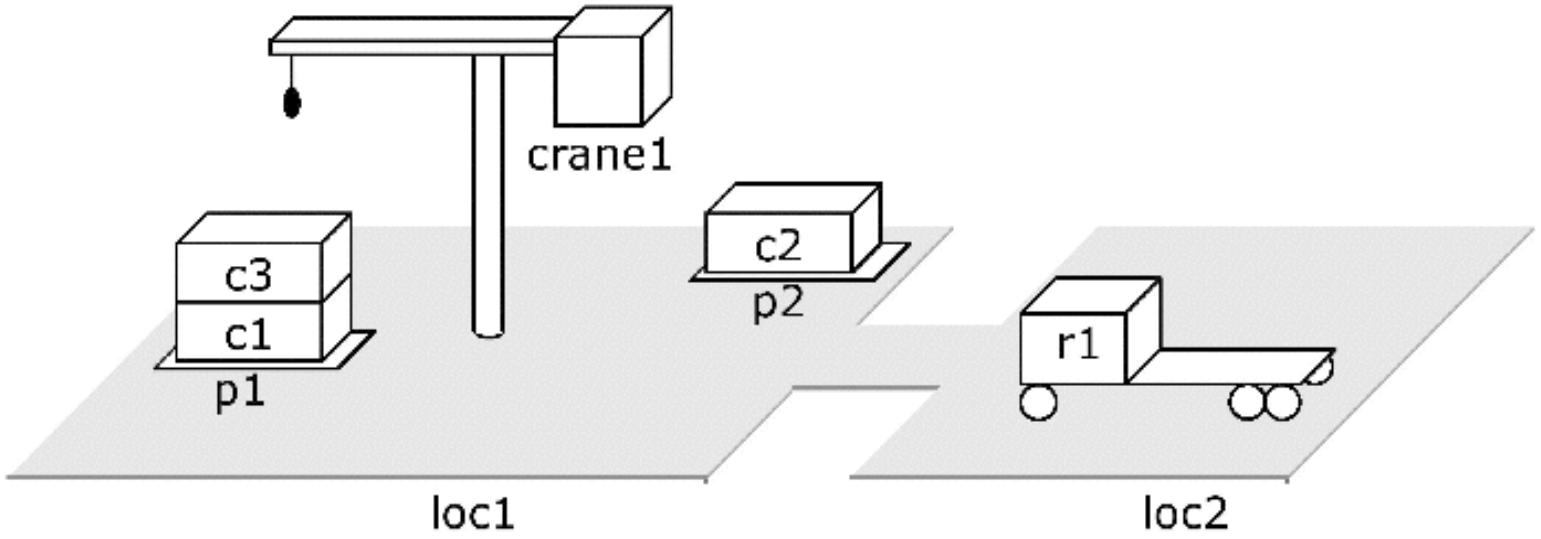
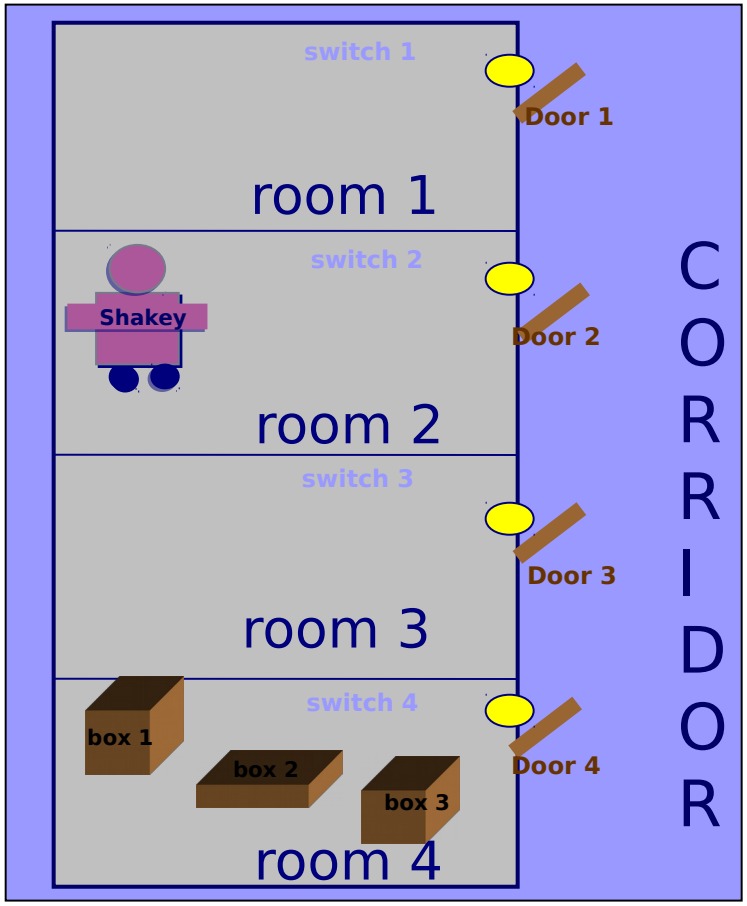


Figure 2.2: The DWR state s_1={attached(p1,loc1), in(c1,p1), in(c3,p1), top(c3,p1), on(c3,c1), on(c1,pallet), attached(p2,loc1), in(c2,p2), top(c2,p2), on(c2,pallet), belong(crane1,loc1), empty(crane1), adjacent(loc1,loc2), adjacent(loc2,loc1), at(r1,loc2), occupied(loc2), unloaded(r1)}.

Example: Shakey's world



Initial state

Shakey is a robot that can move between several rooms, push objects, climb rigid objects, turn on and turn off the lights.

PDDL programming language

Description of the domain

```
(define (domain DOMAIN_NAME)
  (:requirements [:strips] [:equality] [:typing] [:adl])
  (:predicates (PREDICATE_1_NAME [?A1 ?A2 ... ?AN])
               (PREDICATE_2_NAME [?A1 ?A2 ... ?AN])
              ...)

  (:action ACTION_1_NAME
    [:parameters (?P1 ?P2 ... ?PN)]
    [:precondition PRECOND_FORMULA]
    [:effect EFFECT_FORMULA]
  )

  (:action ACTION_2_NAME
  ...)

...)
```

As there are different levels of expressiveness, each description in PDDL says the necessary requirements. The most common are:
: expressivity strips as in STRIPS
: equality the domain uses the predicate =
: typing the domain defines types of vars.
: adl extended expressiveness:
1) disjunctions and quantifiers in preconditions and objectives,
2) Quantified and conditional effects

PDDL programming language

Description of the problem

```
(define (problem PROBLEM_NAME)
(:domain DOMAIN_NAME)
(:objects OBJ1 OBJ2 ... OBJ_N)
(:init ATOM1 ATOM2 ... ATOM_N)
(:goal CONDITION_FORMULA) )
```

PDDL programming language

Description of the domain

```
(define (domain driverlog)
  (:requirements :strips :typing)
  (:types location locatable - object
          driver truck obj - locatable
  )
  (:predicates
    (at ?obj - locatable ?loc - location)
    (in ?obj1 - obj ?obj - truck)
    (driving ?d - driver ?v - truck)
    (link ?x ?y - location) (path ?x ?y - location)
    (empty ?v - truck)
  )
  (:action LOAD-TRUCK
    :parameters
      (?obj - obj
       ?truck - truck
       ?loc - location)
    :precondition
      (and (at ?truck ?loc) (at ?obj ?loc))
    :effect
      (and (not (at ?obj ?loc)) (in ?obj ?truck)))
```

Description of the problem

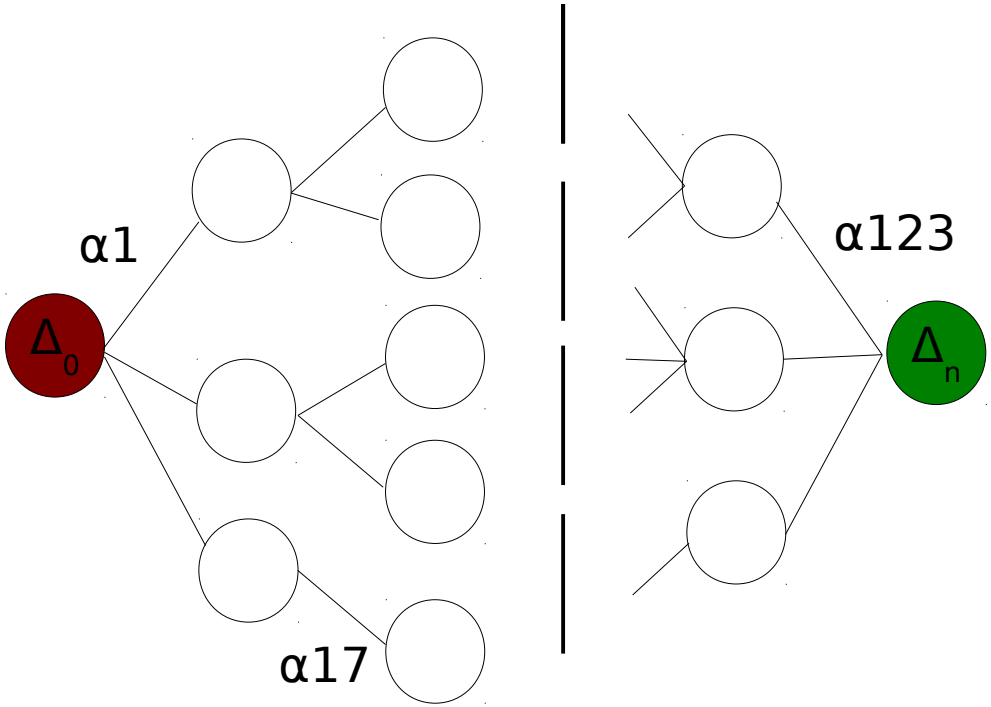
```
(define (problem DLOG-2-2-2)
  (:domain driverlog)
  (:objects
    driver1 - driver
    truck1 - truck
    package1 - obj
    s0 - location
    s1 - location ...)
  (:init
    (at driver1 s12)
    (at truck1 s0)
    (empty truck1)
    (at package1 s0)
    (path s1 p1-0)
    (path p1-0 s1)
    ...
    (link s0 s1)
    (link s1 s0)
    ... )
  (:goal (and (at driver1 s1)
              (at truck1 s1)
              (at package1 s0)
  )))
```

Classic Automatic Planning

- Most relevant approaches:
  - State-space planning
  - Planning in the space of plans (Plan-space planning or PSP)
  - Hierarchical planning (Hierarchical Task Network Planning or HTN)
- Other interesting results
  - Reuse of Plans
  - Domain-specific planning

State-space

- Each node represents a state of the world
- The states are defined by a set of predicates and variables
- A plan is a path within the state space



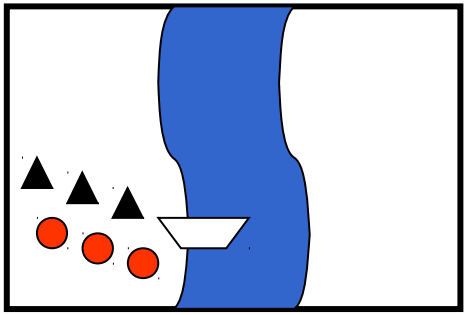
State space: Missionaries and cannibals

Initial state:

- the missionaries, the cannibals, and the ship are on the left bank

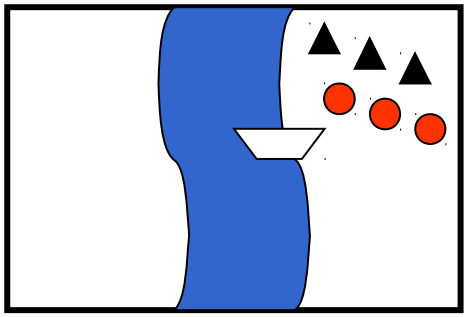
5 possible actions:

- Cross a missionary
- Cross a cannibal
- Two missionaries cross
- They cross two cannibals
- Cross a missionary and a cannibal

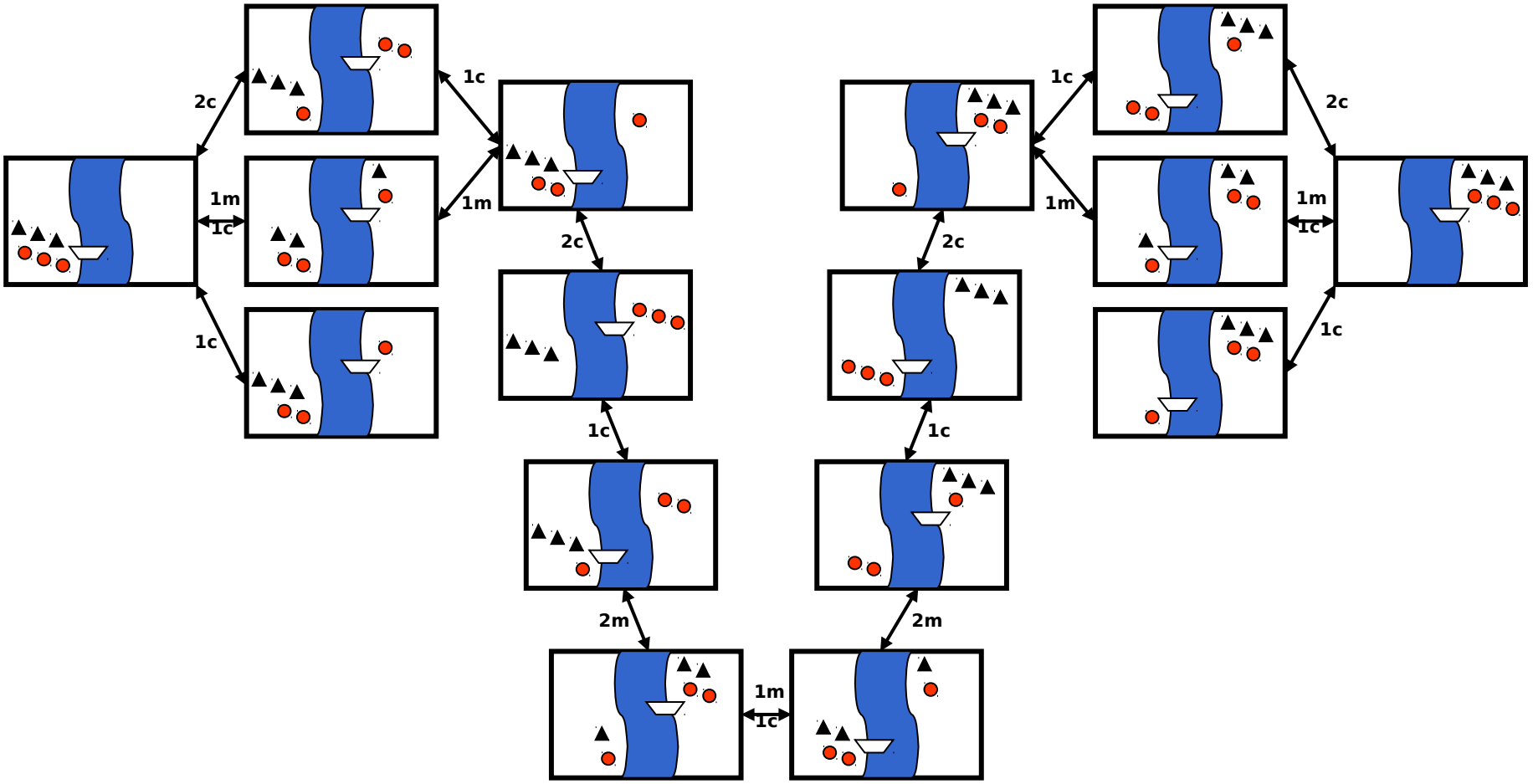


State space: Missionaries and cannibals

- Target state:
  - the missionaries, the cannibals, and the ship are on the right bank
- Solution cost:
  - Cost per edge: 1 for each crossing
  - Path cost: number of crossings = path length
- Path solution:
  - Four optimal solutions
  - Cost = 11

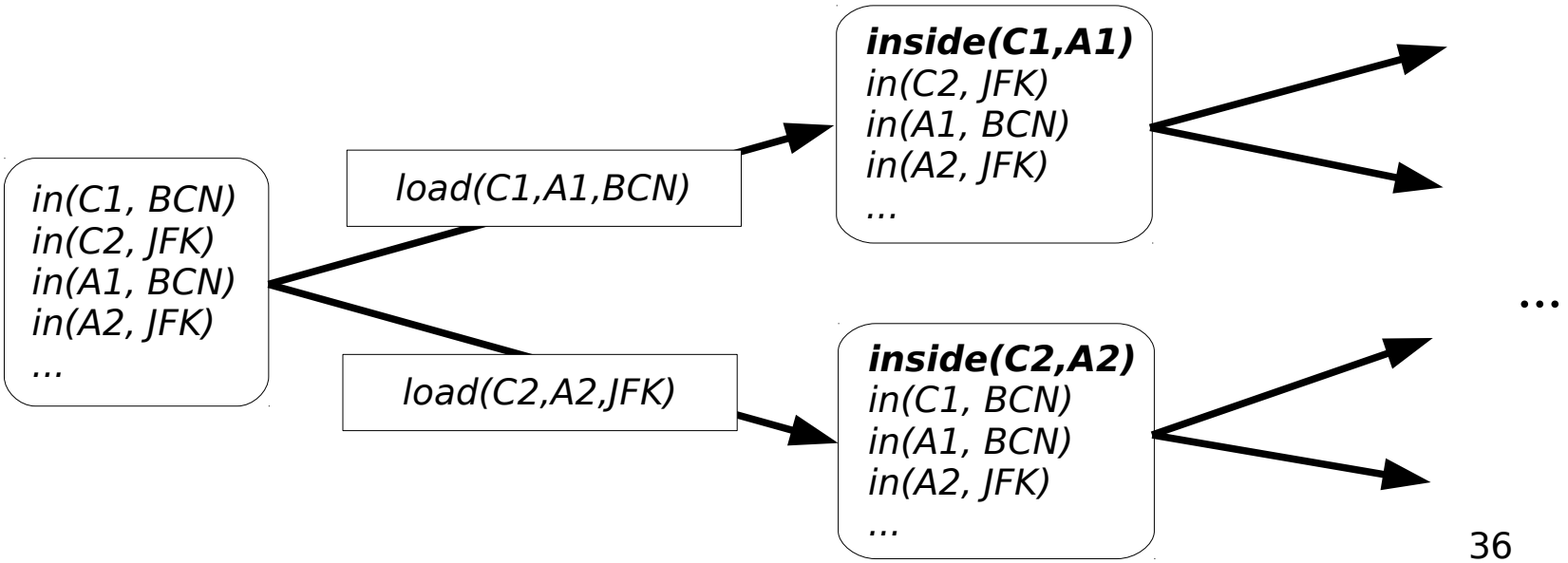


State space: Missionaries and cannibals



State space strategies

- Forward search (progressive planning)
  - The initial state of the search is the initial state of the problem
  - At every moment we try to unify the preconditions with the actions
  - The description of the state is changed by adding or removing literals from the effects of the actions



Deterministic progressive planning

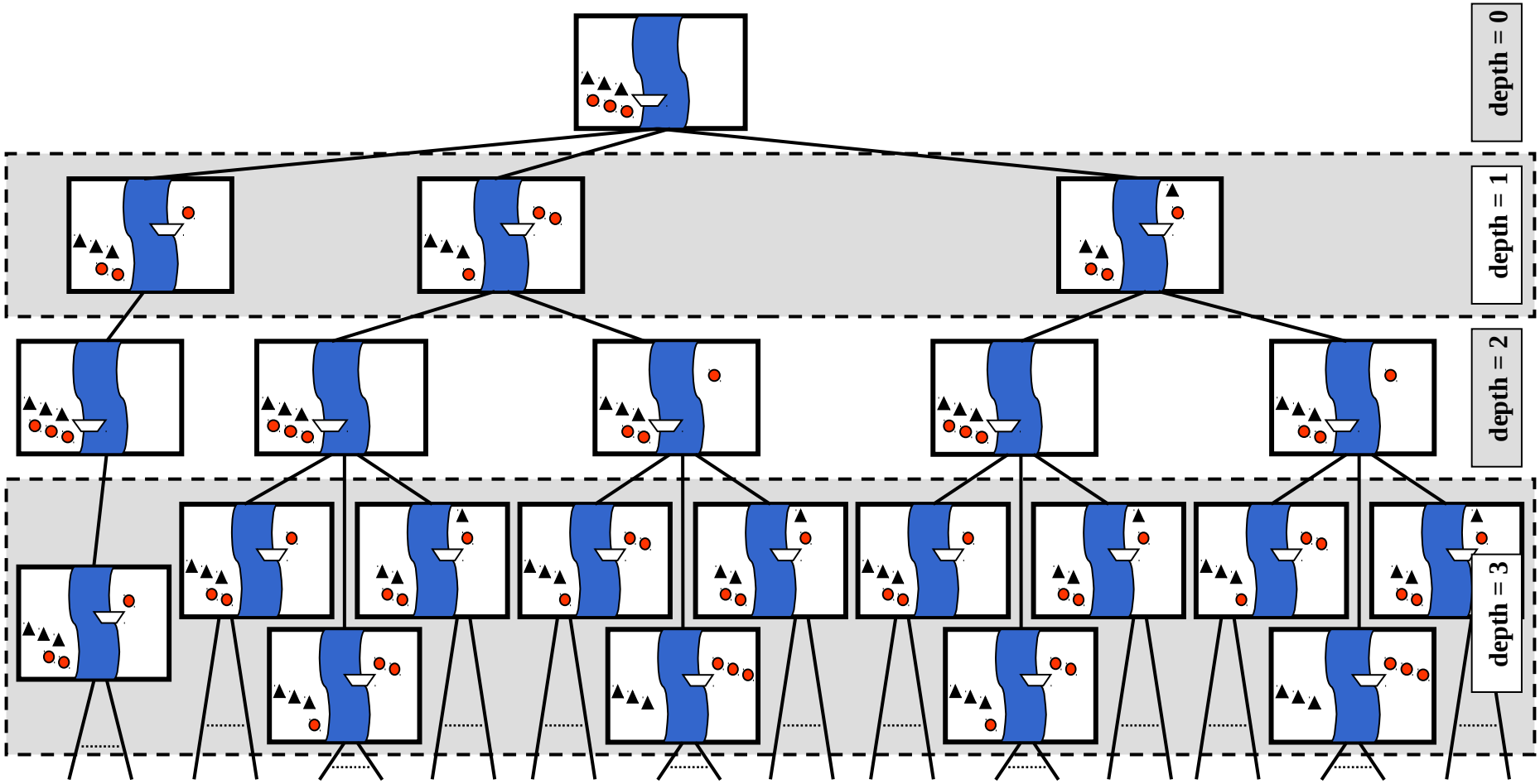
- Deterministic forward search:
  - breadth-first search
  - depth-first search
  - best-first search (ej.: A*)
  - greedy best first
- breadth-first search and best first are complete ...
  - ... but they are not usually practical because they need too much memory (exponential in the length of the solution)
- In practice, depth-first or greedy is often used
  - Problem: they are not complete
  - But classical planning has a finite set of states
  - Depth-first search can be made complete by controlling the cycles

Deterministic progressive planning

- Problem: When the branching factor is very high:
  - There are many applicable actions that do not lead us to the objective
  - Deterministic implementations can waste a lot of time trying multiple irrelevant actions

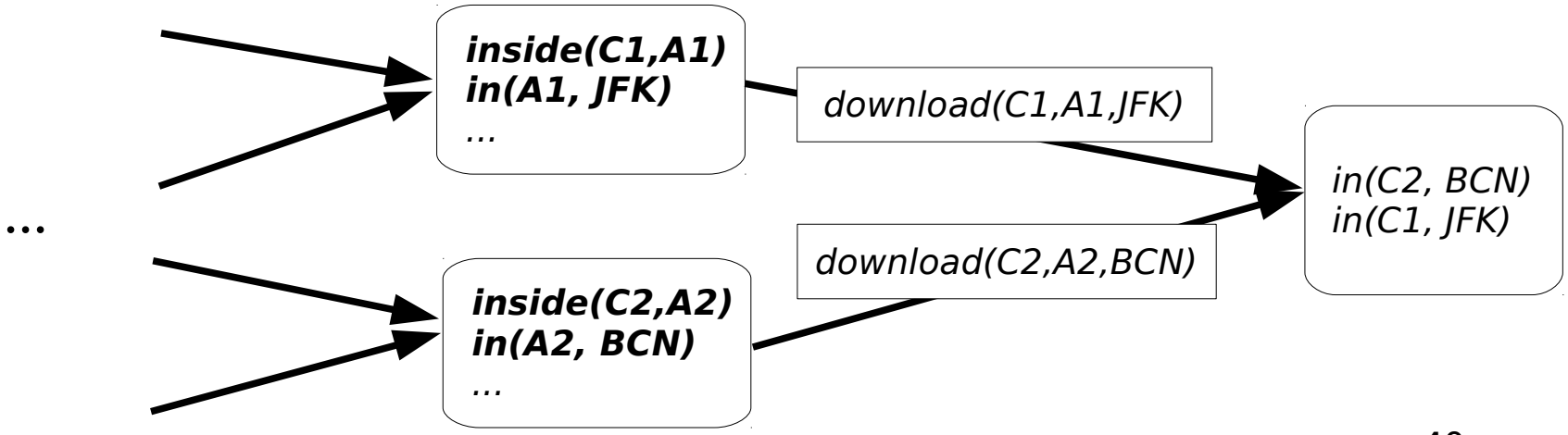
- One possible solution: add domain-specific heuristics

State space: Missionaries and cannibals



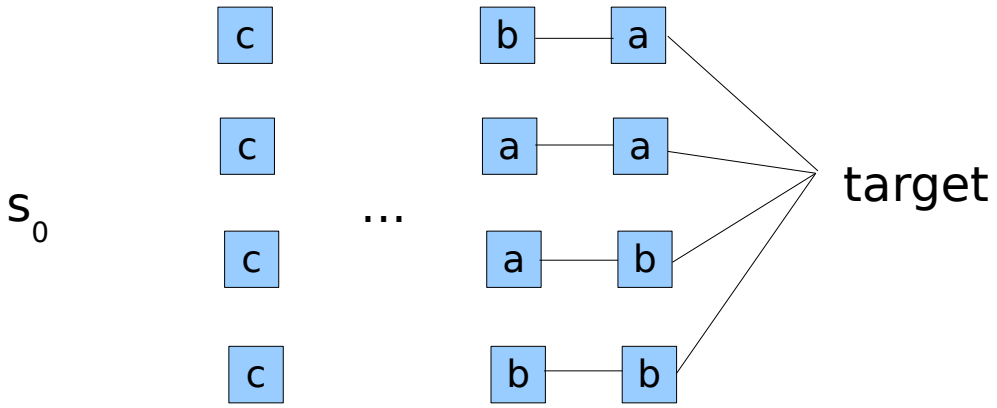
State space strategies

- Backward search (backward planning)
  - The initial state of the search is the final state of the problem
  - At every moment, we try to unify with the effects of the actions. Positive effects are removed from the description.
  - The precondition literals are added except if they already appear in the current description
  - The search ends when all preconditions are satisfied by the initial state of the problem



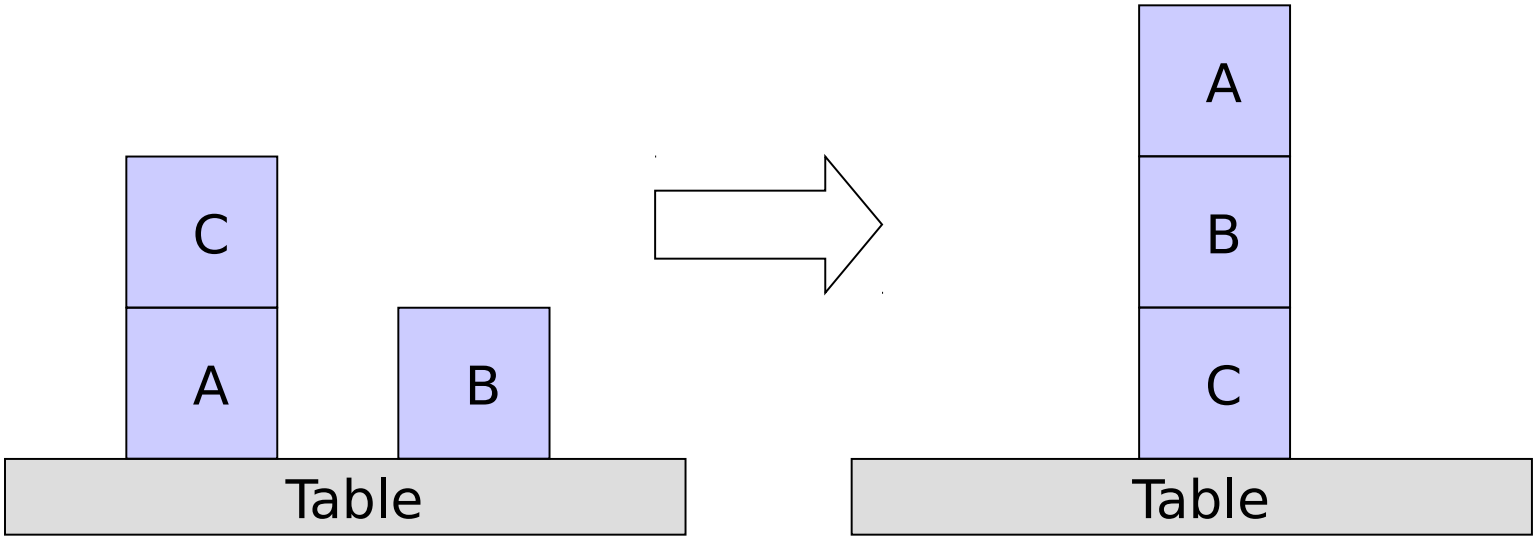
State space strategies

- Problem of regressive planning
  - Although it generates a somewhat smaller search space, it can still be very large and somewhat inefficient
- Example:
  - In the case of three independent actions *a* and *b*, an action *c* that must always precede them, and that there is no path from s0 to the necessary state as input of *c*
  - The algorithm tries every possible order of *a* and *b* before realizing that there is no solution.



STRIPS problems

- STRIPS is not complete:
  - STRIPS cannot find a solution for some problems. For example, by exchanging the values of two variables
  - STRIPS cannot find the optimal solution in others, for example, Sussman anomaly (1975):



{on(A,B), on(B,C)}

STRIPS problems

- Interlaced plans for an optimal solution:
  - Shorter solution to achieve on(A, B):
    - move C from A on the table
    - move A over B
  - Shorter solution to achieve on(B, C):
    - move B over C
  - Shorter solution to achieve on(A, B) and on(B, C):
    - move C from A on the table
    - move B over C
    - move A over B
  - The optimal solution cannot be found by the STRIPS algorithm because:
    - STRIPS cannot change the sub-target during the search

Advanced Techniques in Artificial Intelligence

Curso 2022-2022

German Rigau

German.rigau@ehu.eus

Grado en Ingeniería en Informática