Classical Planning and Scheduling

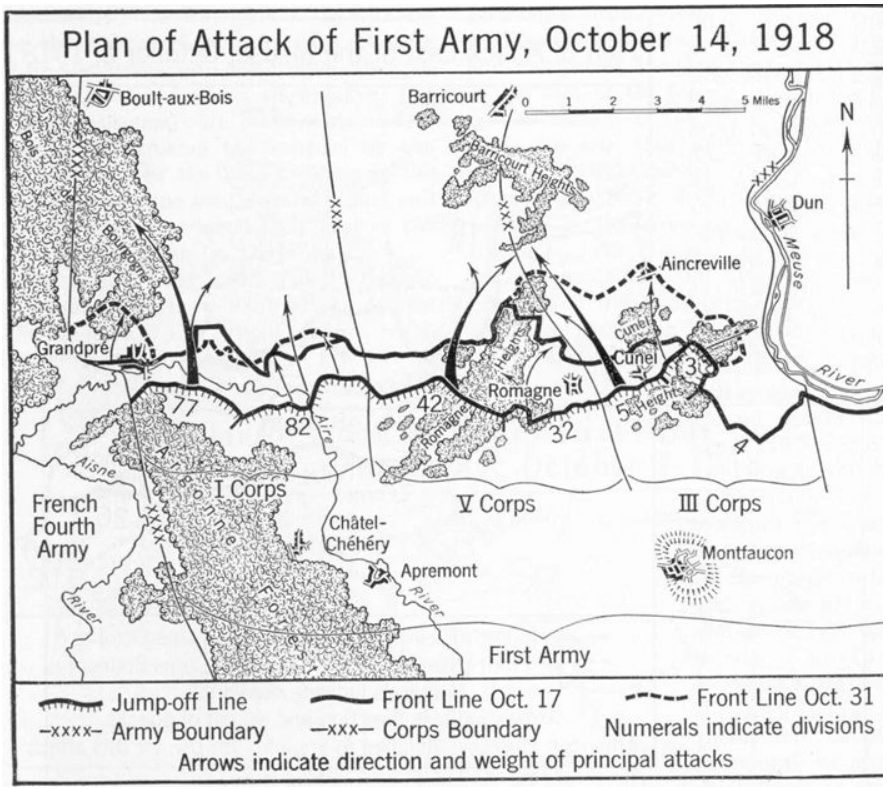
Radek Mařík

CVUT FEL, K13133

16. dubna 2013

Content

Concept of Plan [Nau09]



Plan

- many definitions and aspects ....
- A scheme, program, or method worked out beforehand for the accomplishment of an objective.

Planning and Scheduling [Nau09]

- **Scheduling** . . . assigns in time resources to separate processes,
- **Planning** . . . considers possible interaction among components of plan.

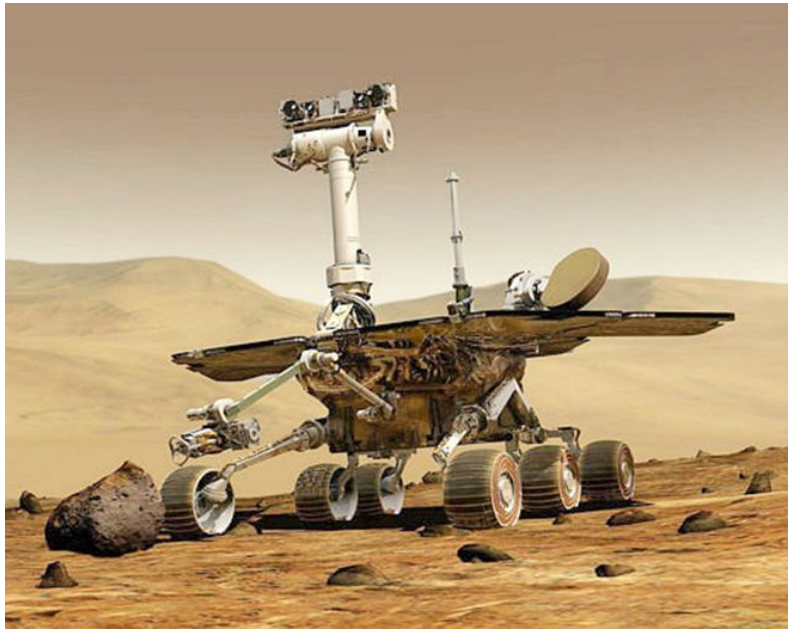
Planning

- Given: the initial state, goal state, operators.
- Find a sequence of operators that will reach the goal state from the initial state
  - Select appropriate actions, arrange the actions and consider the causalities

Scheduling

- Given: resources, actions and constraints.
- Form an appropriate schedule that meets the constraints
  - Arrange the actions, assign resources and satisfy the constrains.

Real Applications - Space Exploration [Nau09]



Projects

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

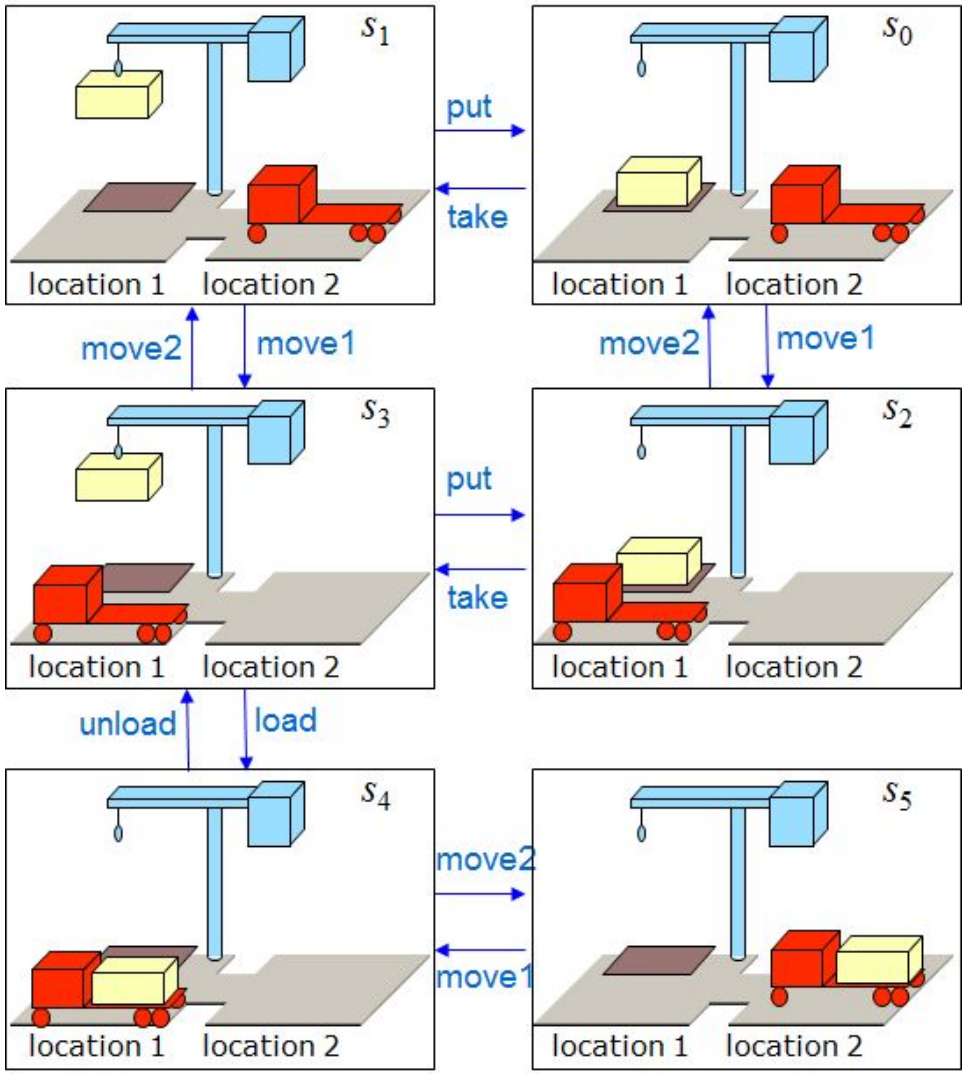
The Dock-Worker Robots (DWR) Domain [Wic11]

Planning procedure illustration

- harbour with several locations (docks),
- docked ships,
- storage areas for containers,
- parking areas for
  - trains,
  - trucks
- Goal:
  - cranes to load and unload ships.
  - robot carts to move containers around

Port of Hamburg

State Transition System Example [Nau09]



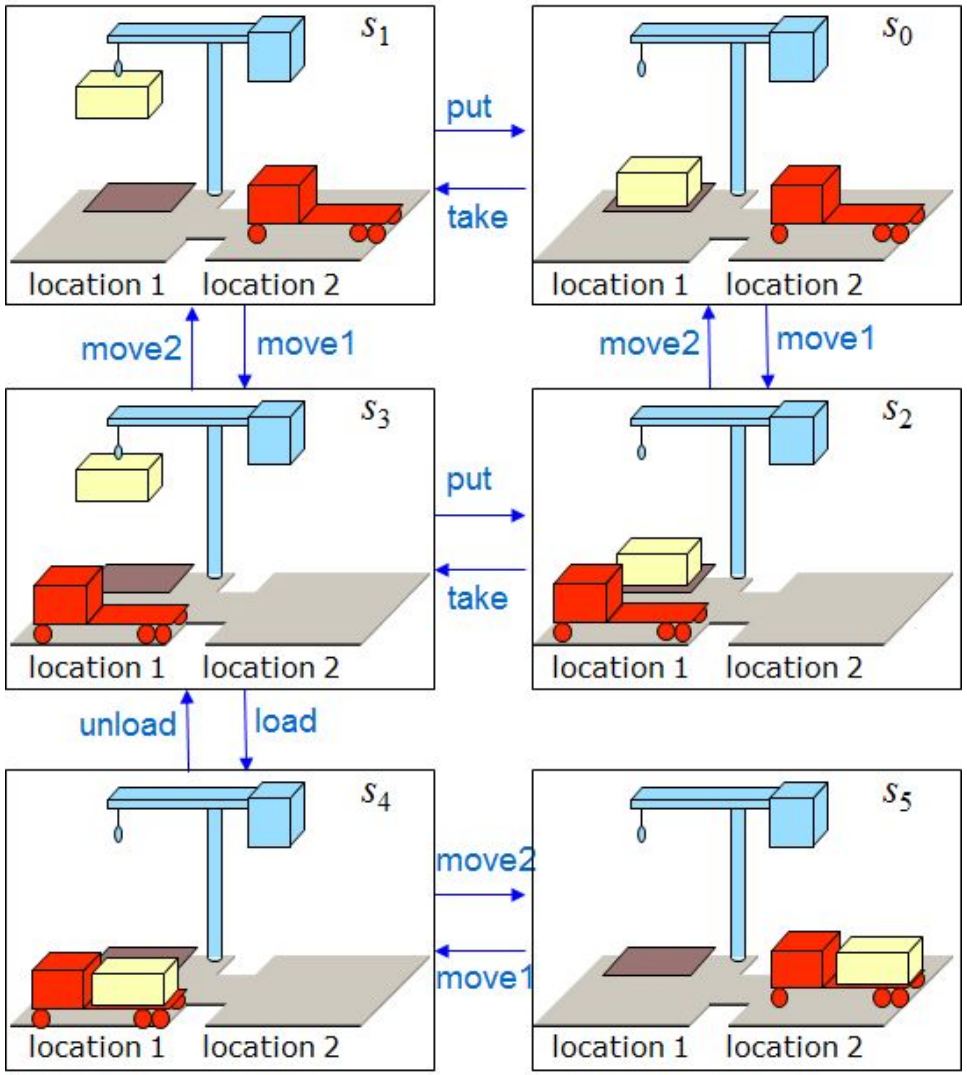
State Trainsition System

- $\Sigma = (S, A, E, \gamma)$
- $S = \{\text{states}\}$
- $A = \{\text{actions}\}$
- $E = \{\text{exogenous events}\}$
- $\gamma = S \times (A \cup E) \rightarrow 2^S$

System Instance

- $S = \{s_0, s_1, \ldots, s_5\}$
- $A = \{move_1, move_2, put, take, load, unload\}$
- $E = \{\}$
- $\gamma = \{\text{see arrows}\}$

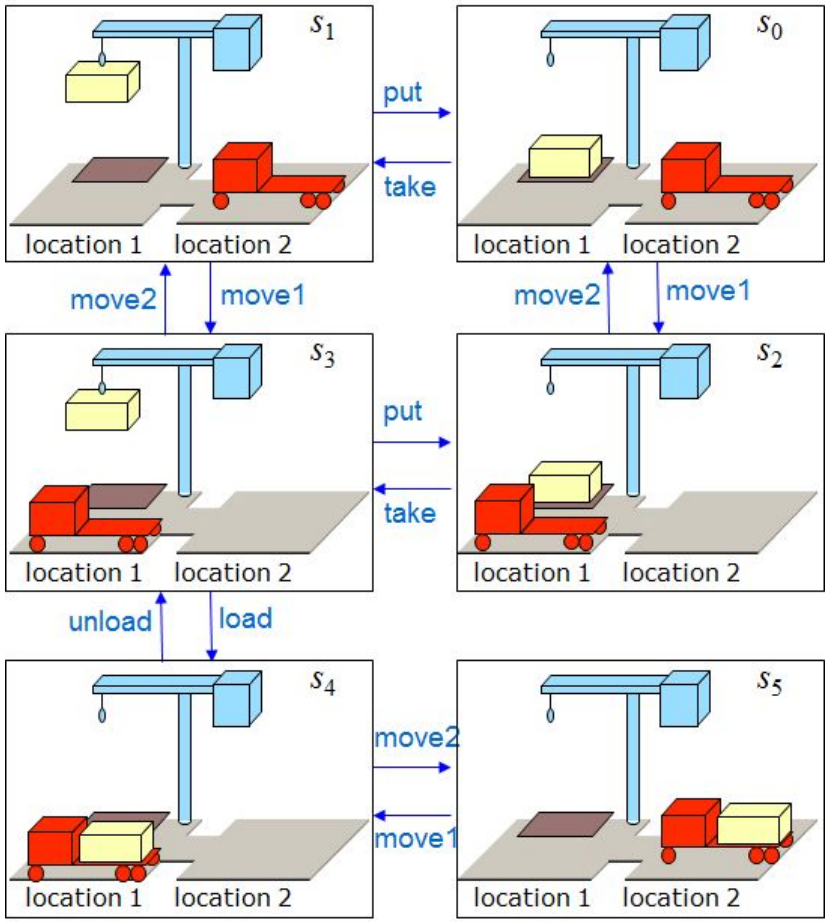
Planning Task [Nau09]



Planning problem

- System description 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 $= s_5$

Plan [Nau09]



Classical plan

- a sequence of actions
- $< take, move_1, load, move_2 >$

Policy:

- partial function from S into A
- $\{(s_0, take), (s_1, move_1), (s_3, load), (s_4, move_2)\}$

Classical Planning - example [Nau09]

Cargo Transportation by Planes

- 10 airports
- 50 aircrafts
- 200 pieces of cargo
- number of states $10^{50} \times (50 + 10)^{200} \approx 10^{405}$
- minimum number of actions $50 \times 9 = 450$
  all cargo located on airpots with no planes
- maximum number of actions $50^{200} \times 9 \approx 10^{340}$
  all cargo and aircrafts in one airport

Reality

- The number of particles in the universe is about 10^{87}

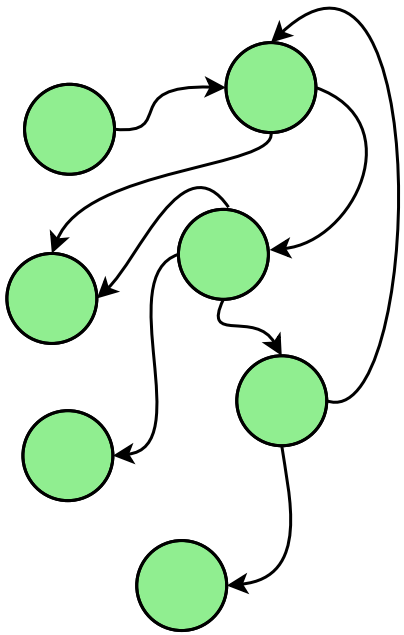
Automated planning research

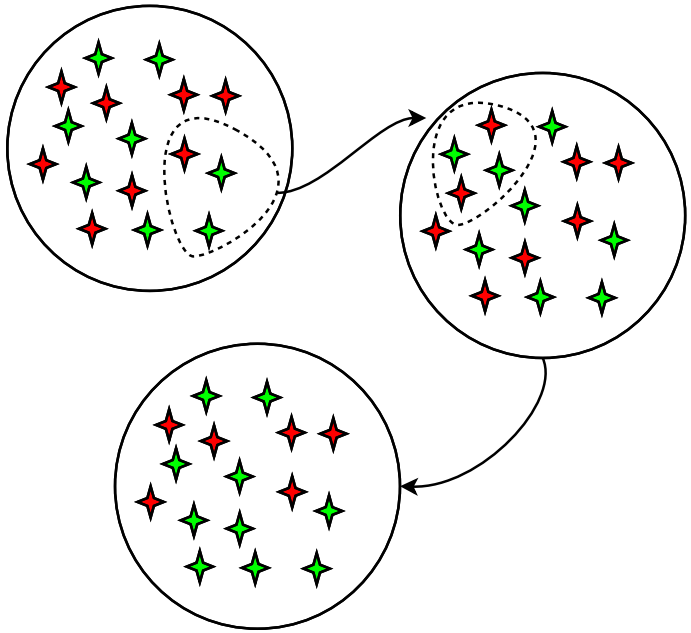
- classical planning mostly
- dozens (hundreds) of different aglorithms

Classical Representatons [Wic11]

- **Planning as Theorem Proving**
  - world state is a set of propositions
  - actions contains applicability conditions as a set of formulas and effects in a form of formulas added or removed if a given action is applied,
- **STRIPS representation**
  - similar to the propositional representation
  - literals of the first order are used instead of propositions
- **a representation using state variables**
  - state is k-tuple of state variables $\{x_1, \ldots, x_k\}$
  - action is a partial function over states

Factored State Representation





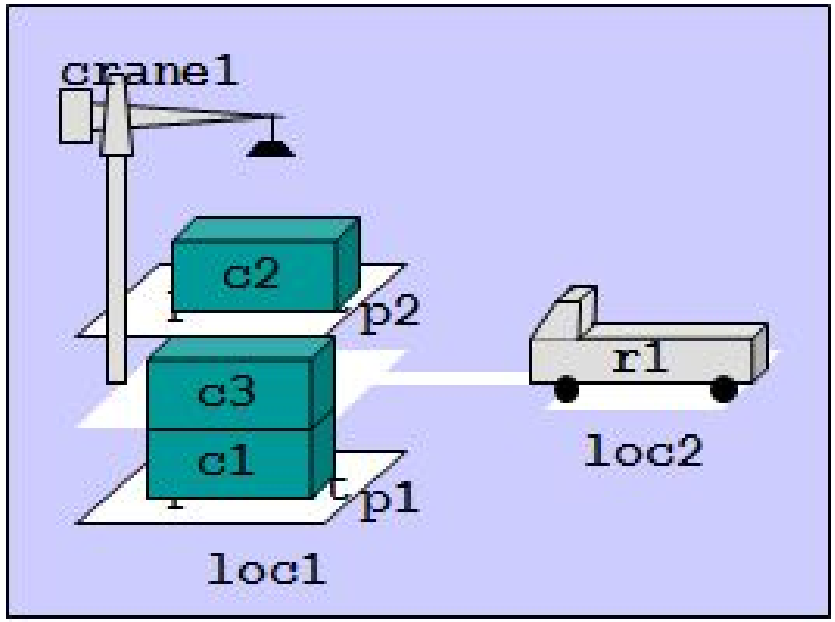
World State Representation

- atomic . . . state is a single indivisible entity
- factored . . . state is a collection of variables

STRIPS State: example [Wic11]

State in DWR Domain

$$
\begin{aligned}
state \quad = \quad & \{attached(p_1, loc_1), \\
& attached(p_2, loc_1), \\
& in(c_1, p_1), in(c_3, p_1), \\
& top(c_3, p_1), on(c_3, c_1), \\
& on(c_1, pallet), in(c_2, p_2), \\
& top(c_2, p_2), on(c_2, pallet), \\
& belong(crane_1, loc_1), \\
& empty(crane_1), \\
& adjacent(loc_1, loc_2), \\
& adjacent(loc_2, loc_1), \\
& at(r_1, loc_2), \\
& occupied(loc_2), \\
& unloaded(r_1)\}
\end{aligned}
$$



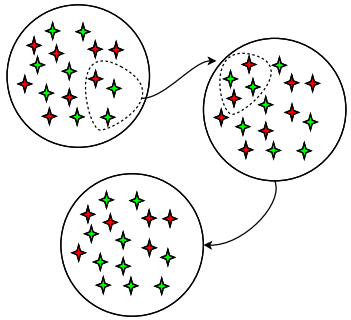
STRIPS - Operator and Action Representations [Wic11]

- **A planning operator** in a STRIPS planning domain is a triple
  - $o = (name(o), precond(o), effects(o))$,
    - the name of the operator $name(o)$
    - is a syntactic expression of the form $n(x_1, \ldots, x_k)$,
    - where n is a (unique) symbol
    - and $x_1, \ldots, x_k$ are all the variables,
    - that appear in o, and
  - the preconditions $precond(o)$ and the effects $effects(o)$ of the operator are sets of literals.
- **An action** in a STRIPS planning domain is a ground instance of a planning operator.

STRIPS Operator: example [Wic11]

- $move(r, l, m)$
  - robot r moves from location l to neighboring location m
  - **precond:** $adjacent(l, m), at(r, l), \neg occupied(m)$
  - **effects:** $at(r, m), occupied(m), \neg occupied(l), \neg at(r, l)$
- $load(k, l, c, r)$
  - crane k in location l loads container c on robot r
  - **precond:** $belong(k, l), holding(k, c), at(r, l), unloaded(r)$
  - **effects:** $empty(k), \neg holding(k, c), loaded(r, c), \neg unloaded(r)$
- $put(k, l, c, d, p)$
  - crane k in location l puts container c onto d in pile p
  - **precond:** $belong(k, l), attached(p, l), holding(k, c), top(d, p)$
  - **effects:**
    $\neg holding(k, c), empty(k), in(c, p), top(c, p), on(c, d), \neg top(d, p)$

Applicability and State Transitions [Wic11]



- Let L be a set of literals
  - L^+ is the set of atoms that are positive literals in L,
  - L^- is the set of all atoms whose negations are in L
- Let a be an action and s a state.
- Then a is **applicable** in $s \Leftrightarrow$:
  - $precond^+(a) \subseteq s$; and
  - $precond^-(a) \cap s == \{\}$
- The state transition function γ for an applicable action a in state s is defined as:
  - $\gamma(s, a) = (s - effects^-(a)) \cup effects^+(a)$

STRIPS: Planning Domain [Wic11]

- Let $\mathcal{L}$ be a function-free first-order language.
- **STRIPS planning domain** on $\mathcal{L}$ is a restricted state-transition system $\Sigma = (S, A, \gamma)$ such that:
  - S is a set of STRIPS states, i.e. sets of ground atoms,
  - A is a set of ground instances of some STRIPSplanning operators O
  - $\gamma : S \times A \to S$ where
    - $\gamma(s, a) = (s - \mathit{effects}^-(a)) \cup \mathit{effects}^+(a)$ if a is applicable in s
    - $\gamma(s, a) =$ undefined otherwise
  - S is closed under γ

STRIPS: Planning Problem [Wic11]

- **A STRIPS planning problem** is a triple $\mathcal{P} = (\Sigma, s_i, g)$ where:
  - $\Sigma = (S, A, \gamma)$ is a STRIPS planning domain on some first-order language $\mathcal{L}$
  - $s_i \in S$ is the initial state
  - g is a set of ground literals describing the goal such that the set of goal states is $S_g = \{s \in S | s \models g\}$

STRIPS Planning Problem: DWR Example [Wic11]

DWR planning problem

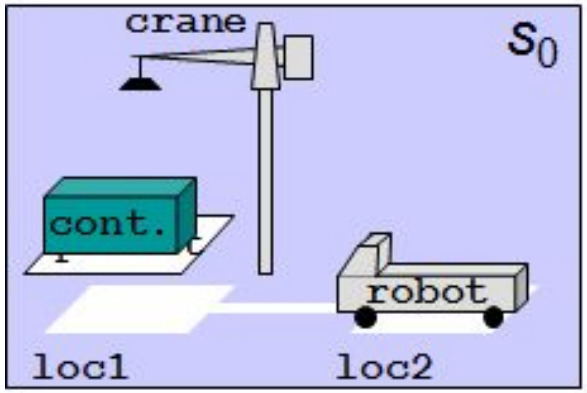
- Σ: STRIPS planning domain DWR
- s_i: any state

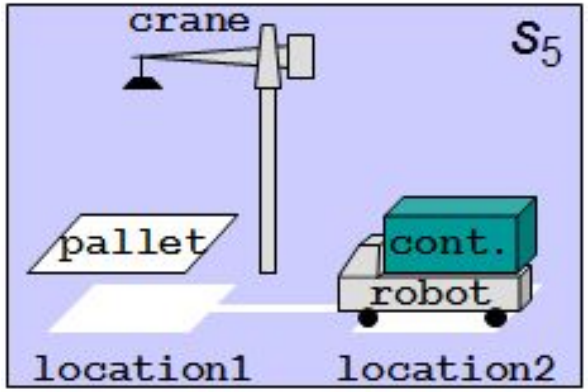
$$
\begin{aligned}
s_0 = \{ & attached(pile, loc_1), \\
& in(cont, pile), top(cont, pile), \\
& on(cont, pallet), belong(crane, loc_1), \\
& empty(crane), adjacent(loc_1, loc_2), \\
& adjacent(loc_2, loc_1), \\
& at(robot, loc_2), \\
& occupied(loc_2), \\
& unloaded(robot)\}
\end{aligned}
$$

- $g \subset L$

$$
\begin{aligned}
g = \{ & \neg unloaded(robot), \\
& at(robot, loc_2)\}
\end{aligned}
$$

tj. $S_g = \{s_5\}$





Overview of PDDL [Wic11]

Planning Domain Definition Language (PDDL)

- http://cs-www.cs.yale.edu/homes/dvm/
- language features (verze 1.x):
  - basic STRIPS-style actions
  - various extensions as explicit requirements
- used to define:
  - planning domains:
    - requirements,
    - types,
    - predicates,
    - possible actions.
  - planning problems:
    - objects,
    - rigid and fluent relations,
    - initial situation,
    - goal description.
- the current version is 3.x

Monkey Planning Domain

```
(define (domain MONKEY)
  (:requirements :strips :typing)
  (:types monkey box location fruit)
  (:predicates
    (isClear ?b - box)                    (onBox ?m - monkey ?b - box)
    (onFloor ?m - monkey)                 (atM ?m - monkey ?loc - location)
    (atB ?b - box    ?loc - location)     (atF ?f - fruit  ?loc - location)
    (hasFruit ?m - monkey ?fruit))
  (:action GOTO
    :parameters (?m - monkey ?loc1 ?loc2 - location)
    :precondition (and (onFloor ?m) (atM ?m ?loc1))
    :effect (and (atM ?m ?loc2) (not (atM ?m ?loc1))))
  (:action PUSH
    :parameters (?m - monkey ?b - box ?loc1 ?loc2 - location)
    :precondition (and (onFloor ?m) (atM ?m ?loc1) (atB ?b ?loc1) (isClear ?b))
    :effect (and (atM ?m ?loc2) (atB ?b ?loc2)
                 (not (atM ?m ?loc1))
                 (not (atB ?b ?loc1))))
  (:action CLIMB
    :parameters (?m - monkey ?b - box ?loc1 - location)
    :precondition (and (onFloor ?m) (atM ?m ?loc1) (atB ?b ?loc1) (isClear ?b))
    :effect (and (onBox ?m ?b) (not (isClear ?b)) (not (onFloor ?m)))  )
  (:action GRAB-FRUIT
    :parameters (?m - monkey ?b - box ?f - fruit ?loc1 - location)
    :precondition (and (onBox ?m ?b) (atB ?b ?loc1) (atF ?f ?loc1))
    :effect (and (hasFruit ?m ?f))))
```

Monkey Planning Problem

```
(define (problem MONKEY1)
  (:domain MONKEY)
  (:objects monkeyJudy - monkey
            bananas - fruit
            boxA - box
            locX locY locZ  - location)
  (:init (and
    (onFloor monkeyJudy)
    (atM monkeyJudy locX)
    (atB boxA locY)
    (atF bananas locZ)
    (isClear boxA)
    )
  )

  (:goal (and (hasFruit monkeyJudy bananas))
)
```

Monkey Planning Problem Solution

```
Begin plan
1 (goto monkeyjudy locx locy)
2 (push monkeyjudy boxa locy locz)
3 (climb monkeyjudy boxa locz)
4 (grab-fruit monkeyjudy boxa bananas locz)
End plan
```

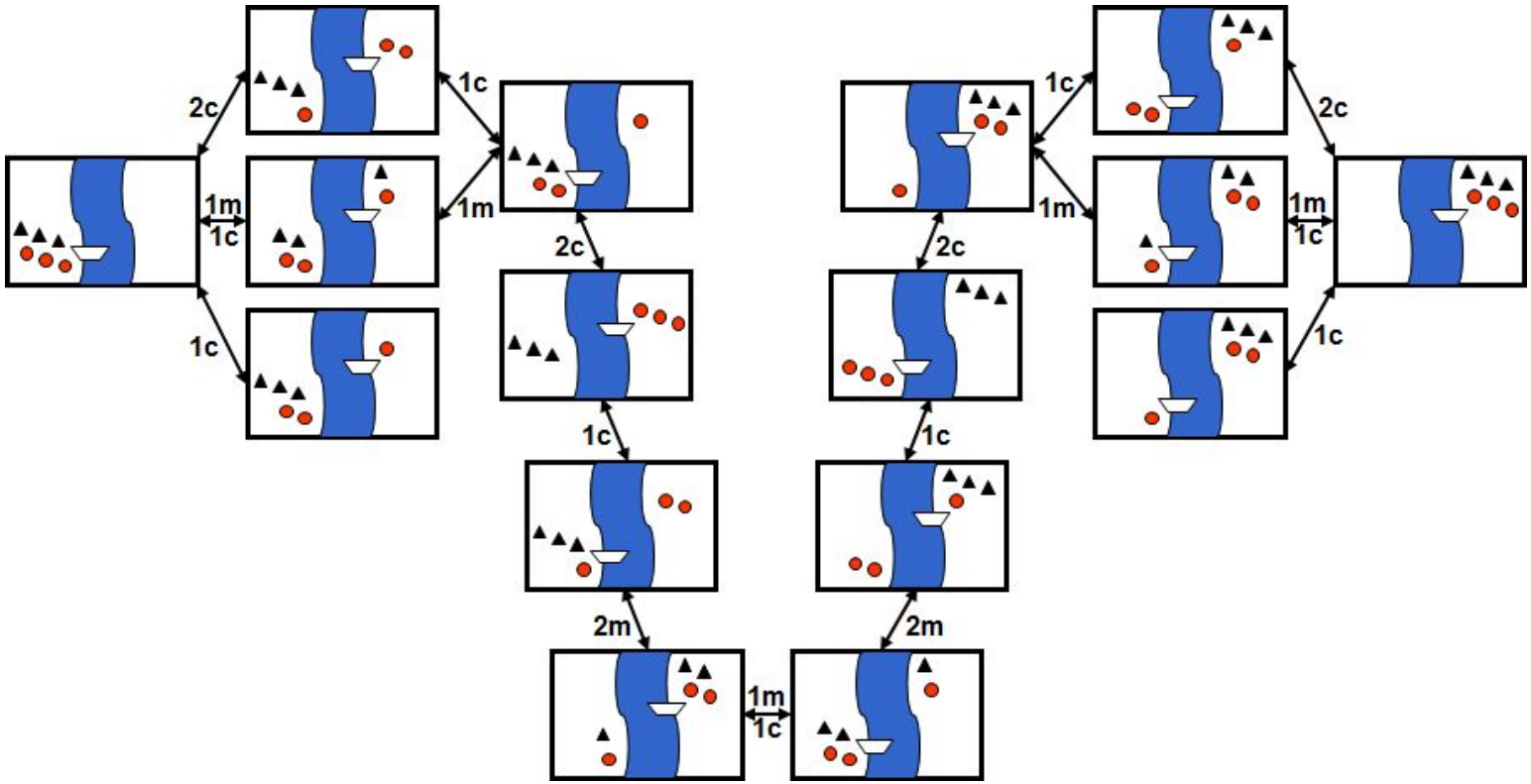
Planning - depth-first search through logical formulea

```
% Depth first search
% ==================
depthFirstSearch(AnsPath) :-
  initialState(Init),
  depthFirst([Init], AnsPath).

depthFirst([S|_], [S]) :-
  finalState(S), !.
depthFirst([S|Path], [S|AnsPath]) :-
  extend([S|Path], S1),
  depthFirst([S1, S |Path], AnsPath).

extend([S|Path], S1) :-
  nextState(S, S1),
  not(memberState(S1, [S|Path])).

memberState(S, Path) :-
  member(S,Path).
```

Planning - a problem specification in Prolog

```
% Farmer, Wolf, Goat, Cabbage
% ===========================
initialState([n,n,n,n]).
finalState([s,s,s,s]).

nextState(S, S1) :-  move(S, S1), safe(S1).

move([F, W, G, C], [F1,  W,  G,  C]) :- cross(F, F1).
move([F, F, G, C], [F1, F1,  G,  C]) :- cross(F, F1).
move([F, W, F, C], [F1,  W, F1,  C]) :- cross(F, F1).
move([F, W, G, F], [F1,  W,  G, F1]) :- cross(F, F1).

safe([F, W, G, C]) :- F=G, !; F=W, F=C.

cross(n,s).
cross(s,n).
%-------
t1(AnsPath) :- depthFirstSearch(AnsPath).
```

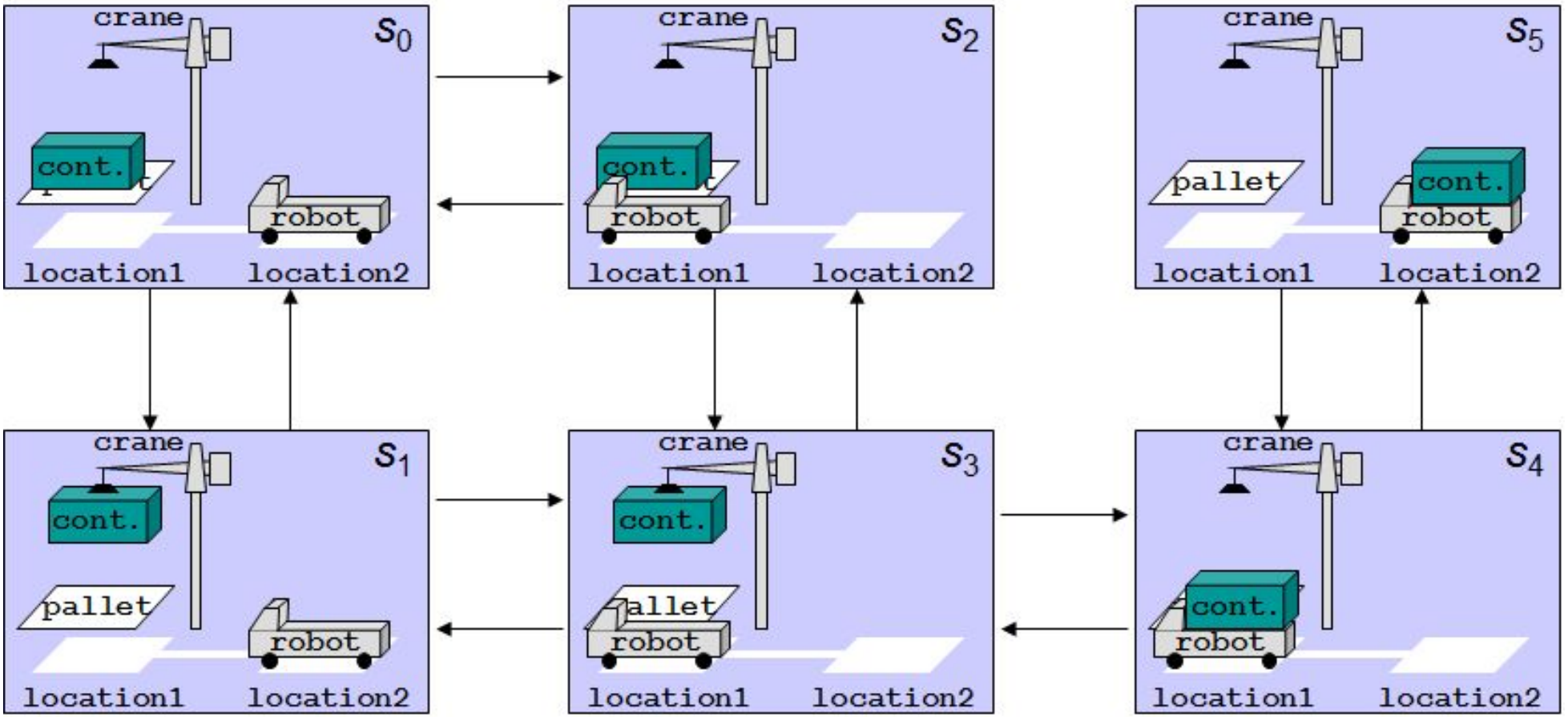
State-Transition Graph Example [Wic11]



Game of Missionaries and Cannibals

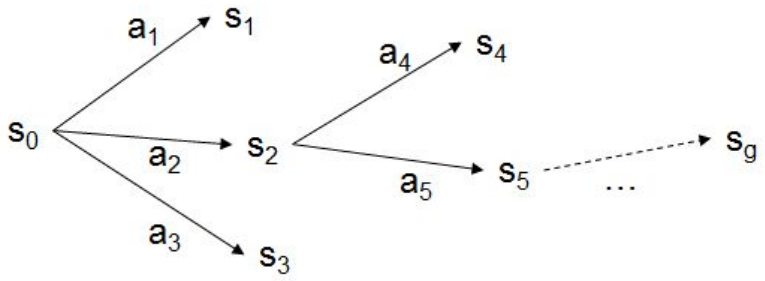
- move 3 cannibals and 3 missionaries across the river
- Whenever the number of cannibals is higher than the number of missionaries somewhere, the missionaries are cooked and eaten.

Planning in State Space - an example [Wic11]



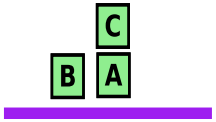
- nodes: closed atoms
- edges: actions (i.e. closed instances of operators)

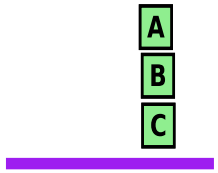
Forward State-Space Search Algorithm [Wic11]



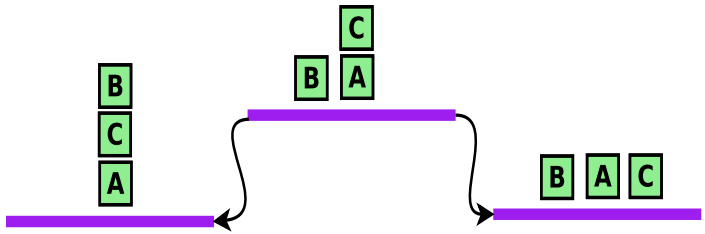
1. Forward-search(O, s_0, g)
   1. $s \leftarrow s_0$
   2. $\pi \leftarrow$ the empty plan
   3. loop
      1. if $s \models g$ then return π
      2. $E \leftarrow \{a | a$ is a ground instance $\in O$ and $precond(a)$ is satisfied in $s\}$
      3. if $E == \emptyset$ then return *FAILURE*
      4. a non-deterministic choice of action $a \in E$
      5. $s \leftarrow \gamma(s, a)$
      6. $\pi \leftarrow \pi.a$

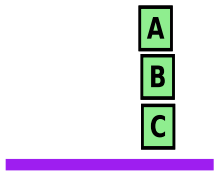
State-Space Searching Example 1 [Wic11]



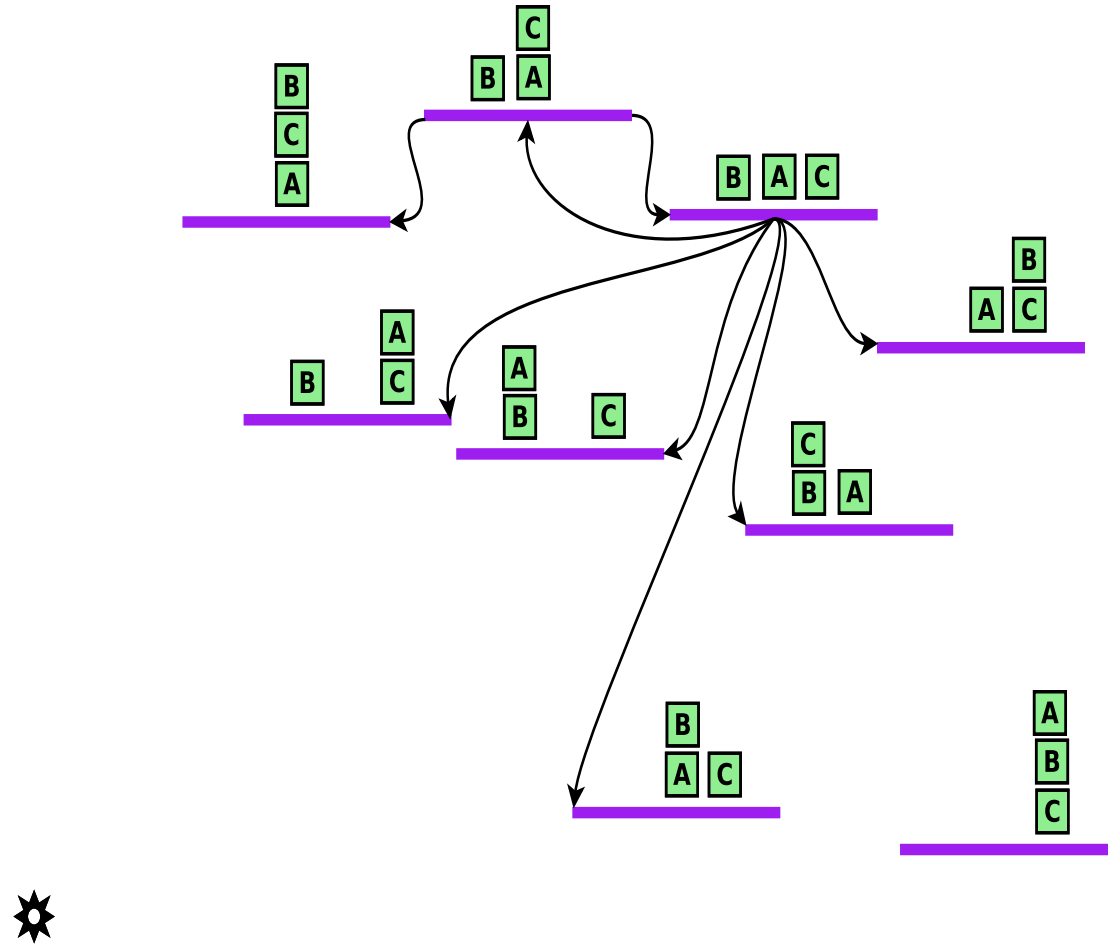


State-Space Searching Example 2 [Wic11]

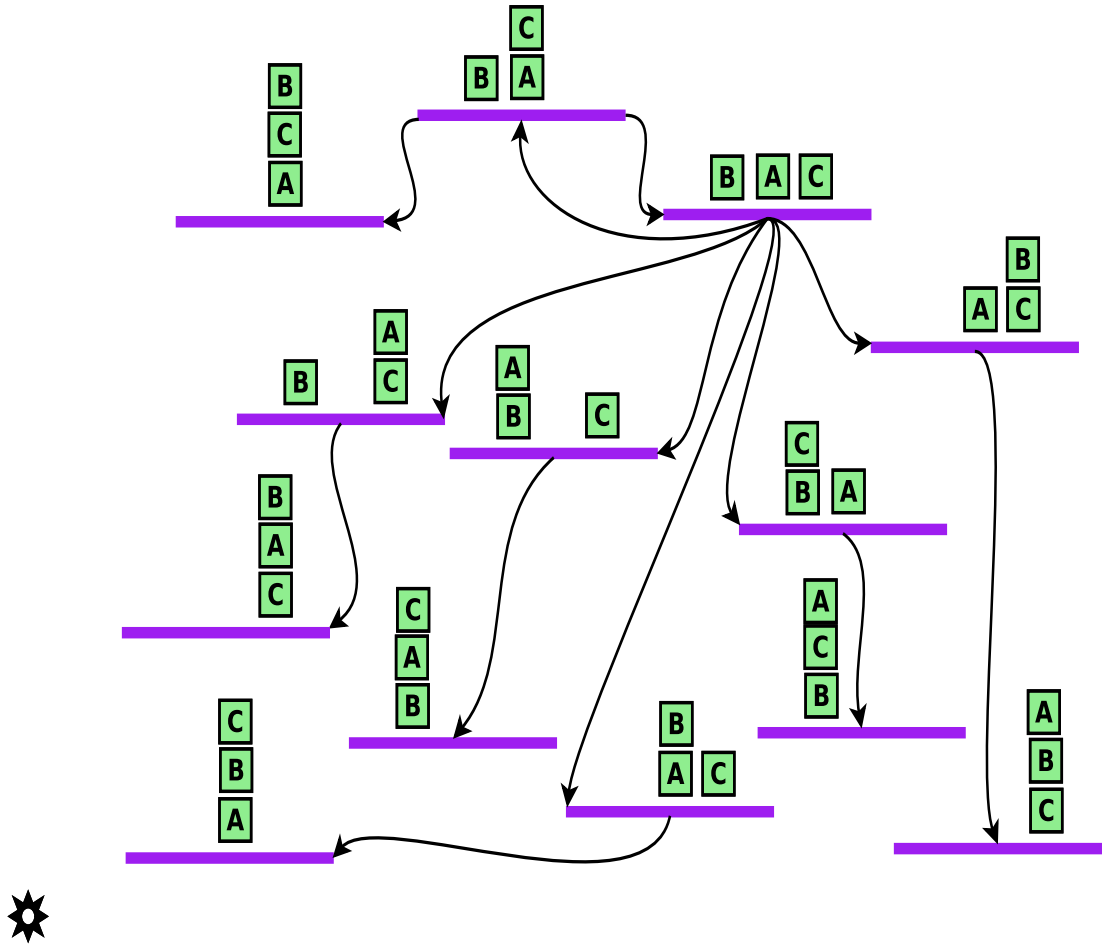




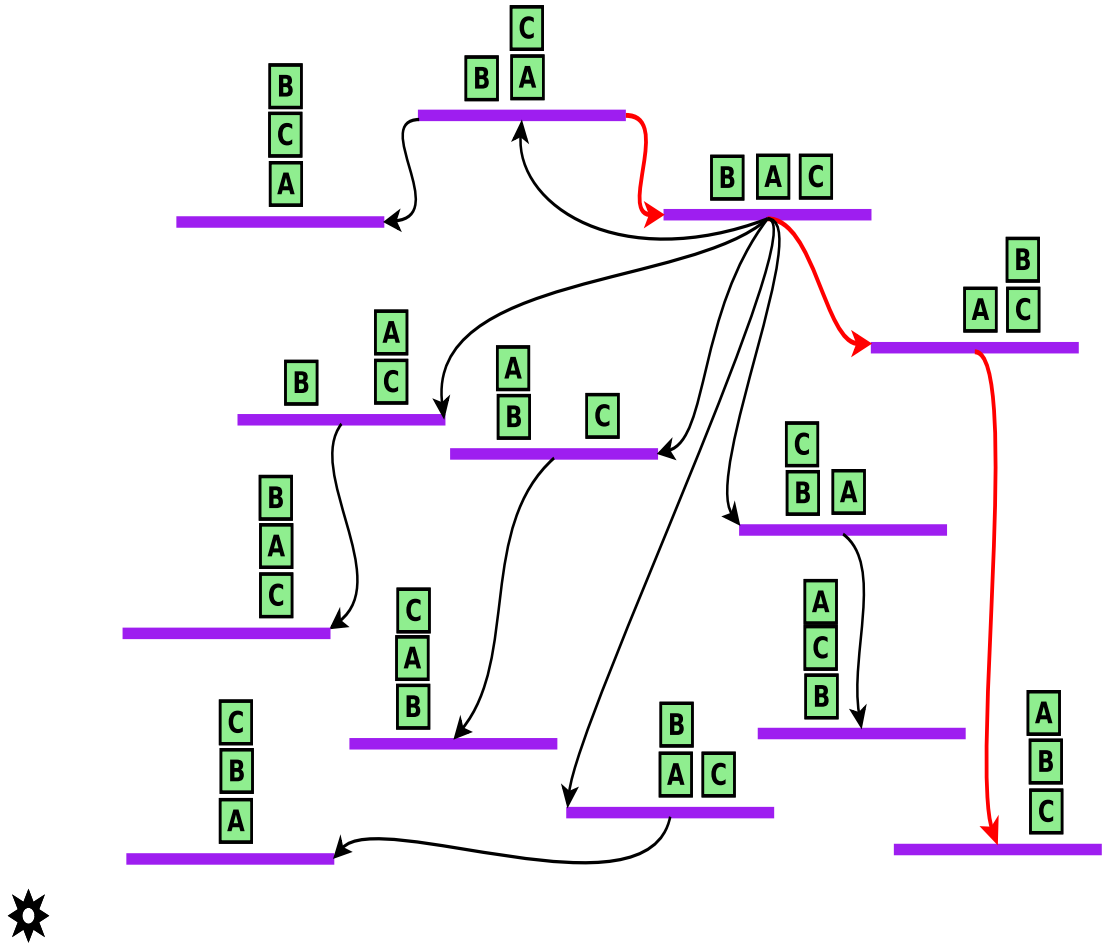
State-Space Searching Example 3 [Wic11]



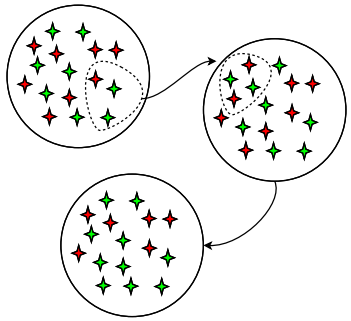
State-Space Searching Example 4 [Wic11]



State-Space Searching Example 5 [Wic11]

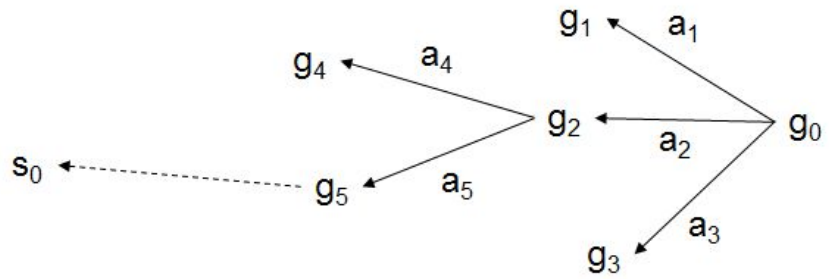


Relevant Actions [Nau09, Wic11]



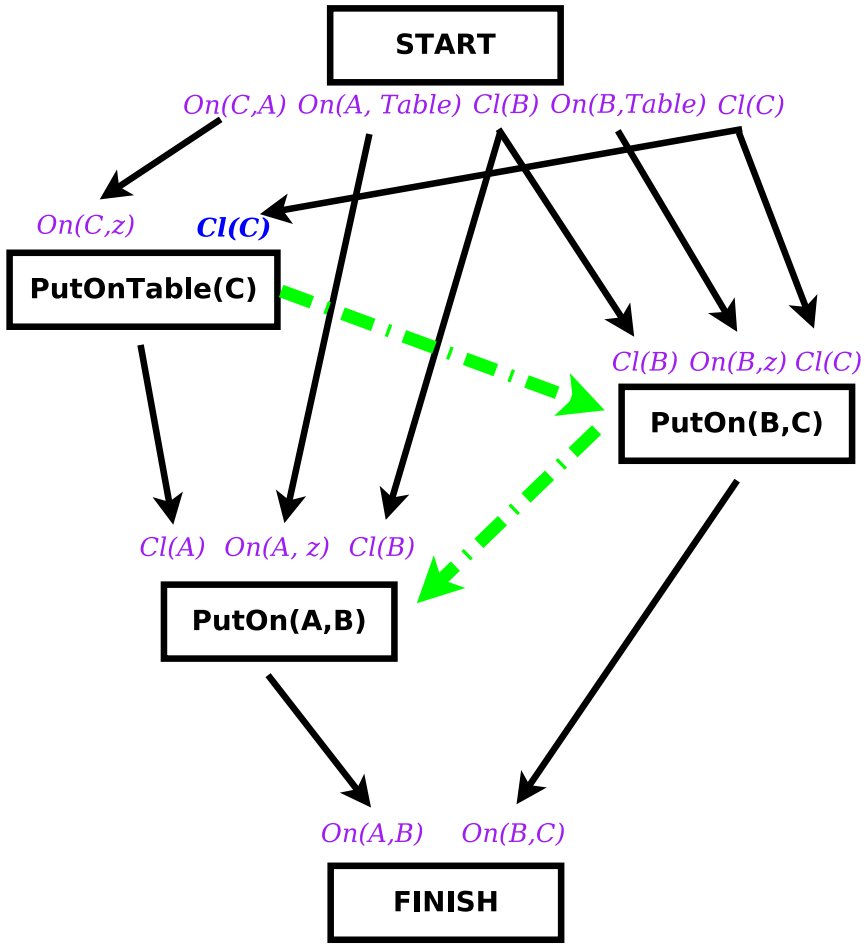
- Let $\mathcal{P} = (\Sigma, s_i, g)$ be a STRIPS planning problem.
- An action a is **relevant** for g if
  - a causes that at least one of literals of g is satisfied
    - $g \cap \mathit{effects}(a) \neq \emptyset$
  - a does not make any of g's literals false
    - $g^+ \cap \mathit{effects}^-(a) = \emptyset \bigwedge g^- \cap \mathit{effects}^+(a) = \emptyset$
- The **Regression Set** of goal g for a relevant action $a \in A$ is:
  - $\gamma^{-1}(g, a) = (g - \mathit{effects}(a)) \cup \mathit{precond}(a)$

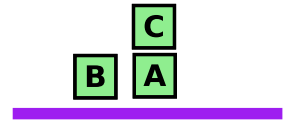
Backward Search [Wic11]

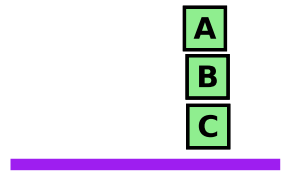


1. Backward-search(O, s_0, g)
   1. $\pi \leftarrow$ the empty plan
   2. loop
      1. if $s_0 \models g$ then return π
      2. $A \leftarrow \{a | a$ is a ground instance of an operator $\in O$ and $\gamma^{-1}(g, a)$ is defined $\}$
      3. if $A == \emptyset$ then return *FAILURE*
      4. nondeterministically choose an action $a \in A$
      5. $\pi \leftarrow a.\pi$
      6. $g \leftarrow \gamma^{-1}(s, a)$

Sussman Anomaly - a Block World Example IV [Nau09]







State-Space vs. Plan-Space Search [Wic11]

state-space search

- search through graph of nodes representing world states

plan-space search

- search through graph of partial plans
- nodes: partially specified plans
- arcs: plan refinement operations
- solutions: partial-order plans
  - temporal ordering of actions
  - rationale: what the action achieves in the plan
  - subset of variable bindings

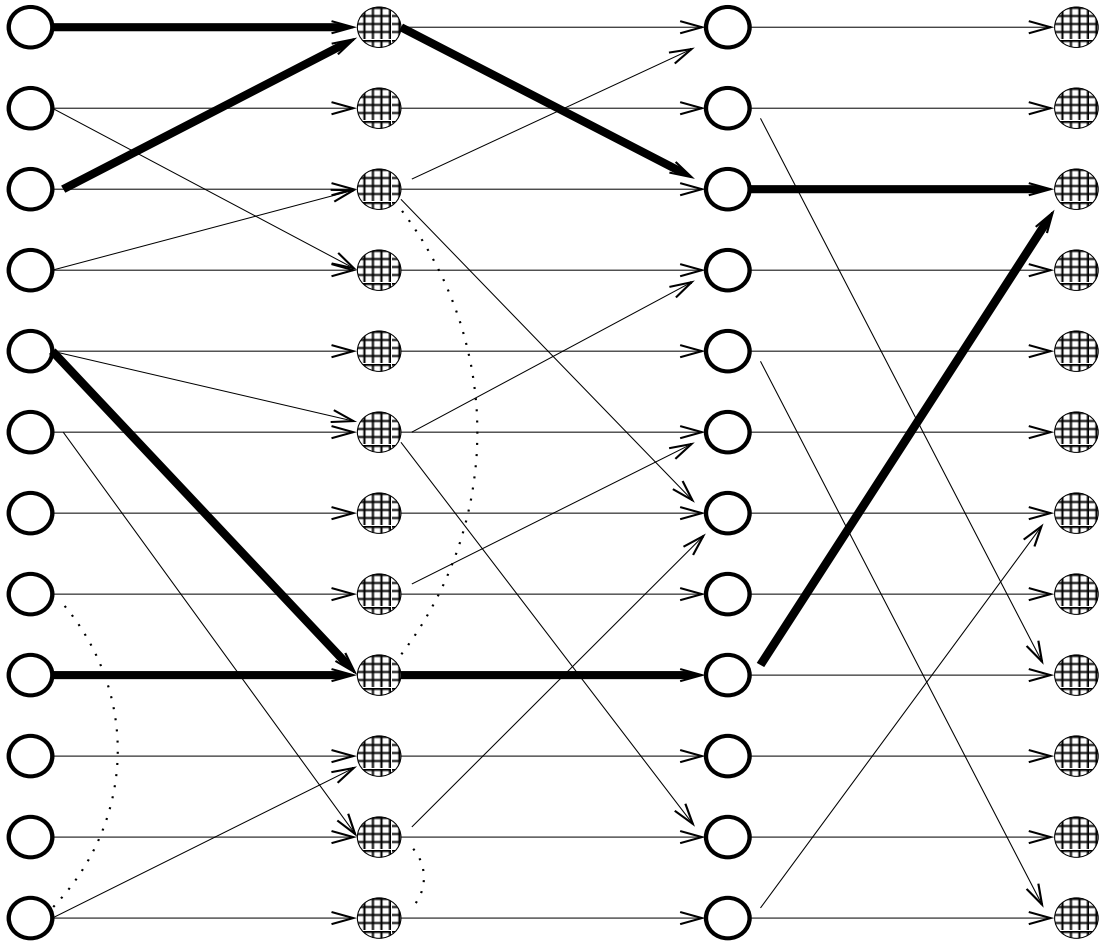
Plan-Space Planning - constraints [Nau09]

- ordering constraints
  - action α must be performed before β ($\alpha \prec \beta$)
- binding constraints
  - inequality constraints, i.e. $v_1 \neq v_2$ or $v_1 \neq c$
  - equality constraints and substitutions, i.e. $v_1 = v_2$ or $v_1 = c$
- causal links
  - use action α to create condition p required by action β

GRAPHPLAN planner

- 1997
- plans are represented as a *planning graph*,
  - the idea is very similar to dynamic programming or network flow solutions
- All plans are constructed concurrently.
  - graph extending (forward run)
  - plan searching (backward run)
- The planner maintains a mutually exclusive relation (*mutex*) between nodes representing applied actions and state propositions.
- The cycling issue is removed.
- Action schemas with parameters cannot be used.
  - It create a huge space of propositions.
- There are many supporting strategies speeding up planning significantly.
- The implementations are capable to create plans with more then 50-100 action calls in minutes.

GraphPlan - Planning Graph



Implementations of planners

Initial attempts

- STRIPS [1971] . . . , the first planner, regressive planning through action preconditions

State/Plan space

- WARPLAN [1973] . . . a linear planner, Sussman anomaly solved using action shifting
- PWEAK, TWEAK [1987], UCPOP [1992] . . . a partial order planner

Planning graphs

- GRAPHPLAN[1997] . . . a breakthrough graphplan planner
- Blackbox [1998] . . . combines GRAPHPLAN and SATPLAN
- FF [2000] . . . a planning graph heuristics with a very fast forward and local search

Time, schedules, and resources [RN10]

- Classical planning representation
  - What to do
  - What order
- Extensions
  - How long an action takes
  - When it occurs
- Scheduling
  - Temporal constraints,
  - Resource contraints.
- Examples
  - Airline scheduling,
  - Which aircraft is assigned to which fligths
  - Departure and arrival time,
  - A number of employees is limited.
  - An aircraft crew, that serves during one flight, cannot be assigned to another flight.

General Approach [Rud13]

Introduction

- Graham's classification of scheduling problems

General solving methods

- Exact solving method
  - Branch and bound methods
- Heuristics
  - dispatching rules
  - beam search
  - local search:
    simulated annealing, tabu search, genetic algorithms
- Mathematical programming: formulation
  - linear programming
  - integer programming
- Constraing satisfaction programming

Schedule [Rud13]

Schedule:

- determined by tasks assignments to given times slots using given resources,
  where the tasks should be performed

Complete schedule:

- all tasks of a given problem are covered by the schedule

Partial schedule:

- some tasks of a given problem are not resolved/assigned

Consistent schedule:

- a schedule in which all constraints are satisfied w.r.t. resource and tasks, e.g.
  - at most one tasks is performed on a signel machine with a unit capacity

Consistent complete schedule vs. consistent partial schedule

Optimal schedule:

- the assigments of tasks to machines is optimal w.r.t. to a given optimization criterion, e.g..
  - min C_{max}: makespan (completion time of the last task) is minimum

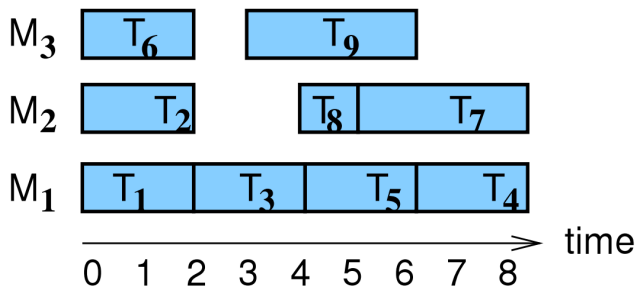
Terminology of Scheduling [Rud13]

Scheduling

concerns optimal allocation or assignment of resources, to a set of tasks or activities over time

- limited amount of resources,
- gain maximization given constraints

- Machine $M_i, i = 1, \ldots, m$
- Jobs $J_j, j = 1, \ldots, n$
- (i, j) **an operation** or processing of jobs j on machine i
  - a job can be composed from several operations,
  - example: job 4 has three operations with non-zero processing time (2,4),(3,4),(6,4), i.e. it is performed on machines 2,3,6

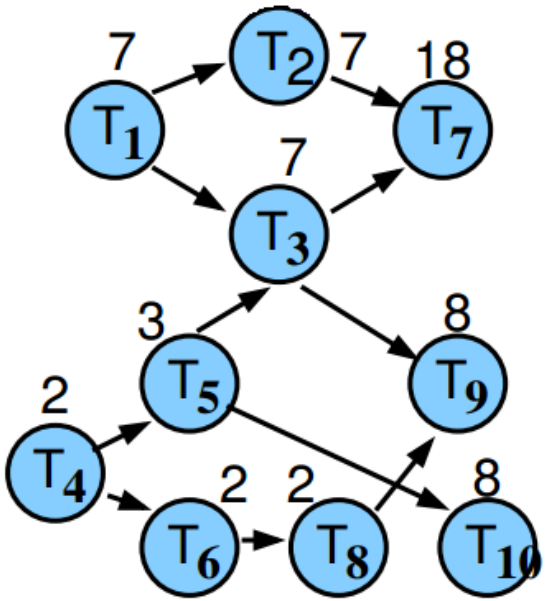


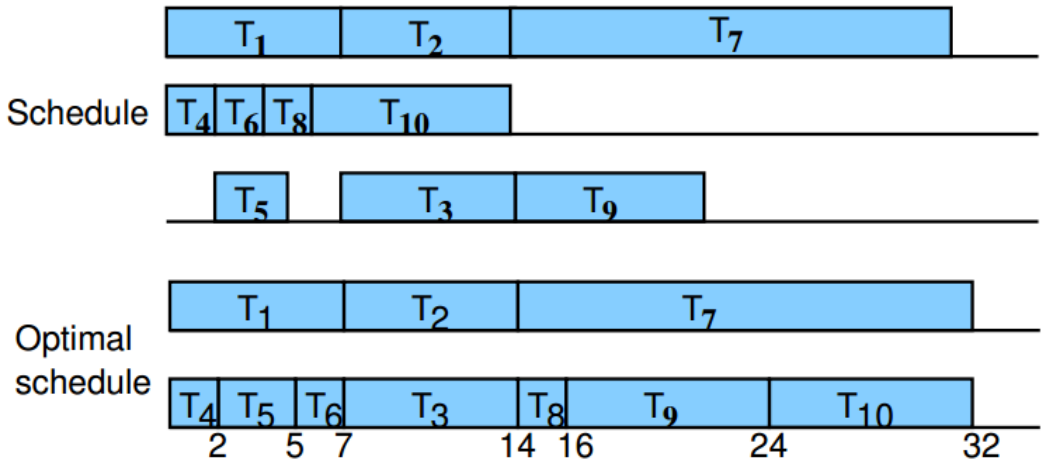
Static and dynamic parameters of jobs [Rud13]

- Static parameters of job
  - **processing time** p_{ij}, p_j:
    processing time of job j on machine i
  - **release date of** j r_j:
    earliest starting time of jobs j
  - **due date** d_j:
    committed completion time of job j (preference)
  - vs. **deadline**:
    time, when job j must be finised at latest (requirement)
  - **weight** w_j:
    importance of job j relatively to other jobs in the system
- Dynamic parameters of job
  - **start time** S_{ij}, S_j:
    time when job j is started on machine i
  - **completion time** C_{ij}, C_j:
    time when job j execution on machine i is finished

Example: bike assembly [Rud13]

- 10 jobs with given processing time
- Precedence constraints
  - a given job can be executed after a specified subset of jobs
- Non-preemptive jobs
  - jobs cannot be interrupted
- Optimization criteria
  - makespan minimization
  - worker number minimization





Scheduling Examples [Rud13]

- Scheduling of semiconductor manufacturing
  - a large amount of heterogenous products,
  - different amounts of produced items,
  - machine setup cost, required processing time guarantees
- Scheduling of supply chains
  - ex. a forest region – paper production – products from paper – distribution centers – end user
  - manufacturing cost, transport, storage minimization,
- Scheduling of paper production
  - input - wood, output - paper roles, expensive machines, different sorts of papers,
  - storage minimization
- Car assembly lines
  - manufacturing of different types of cars with different equipment,
  - throuput optimization, load balancing
- Lemonade filling into bottles
  - 4 flavors, each flavor has its own filling time,
  - cycle time minimization, one machine

Scheduling Examples II [Rud13]

- Scheduling of hospital nurses
  - different numbers of nurses in working days and weekends,
  - weaker requirements for night shift rostering,
  - assignment of nurses to shifts, requirement satisfaction, cost minimization
- Grid computing scheduling
  - clusters, supercomputers, desktops, special devices,
  - scheduling of computation jobs and related resources,
  - scheduling of data transfers and data processing
- University scheduling
  - Time and rooms selection for subject education at universities
  - constraints given for subject placement,
  - preference requirements for time and room optimization,
  - minimization of overlapping subjects for all students,

Scheduling vs. timetabling [Rud13]

Scheduling . . . scheduling/planning

- resource allocation for given constraints over objects placed in time-space so that total cost of given resources is minimized,
- focus is given on object ordering, precedence conditions
  - ex. manufacturing scheduling: operation ordering determination, time dependencies of operation is important,
- **schedule**: specifies space and time information

Timetabling

- resource allocation for given constraints over objects pakce in time-space so that given criteria are met as much as possible,
- focus is given on time placement of objects
- time horizon is often given in advance (a number of scheduled slots)
  - ex. education timetabling: time and a place is assigned to subjects
- **timetable**: shows when and where events are performed.

Sequencing and Rostering [Rud13]

Sequencing

- for given constraints:
  - a construction of job order in which they will be executed
- **sequence**
  - an order in which jobs are executed
- ex. lemonade filling into bottles

Rostering

- resource allocation for given constraints into slots using patterns
- **roster**
  - a list of person names, that determines what jobs are executed and when
- ex. a roster of hospital nurses, a roster of bus drivers

Graham's classification [Rud13, Nie10]

Graham's classification $\alpha|\beta|\gamma$

(Many) Scheduling problems can be described by a three field notation

- α: the machine environment
  - describes a way of job assingments to machines
- β: the job characteristics,
  - describes constraints applied to jobs
- γ: the objective criterion to be minimized
- complexity for combinations of scheduling problems

Examples

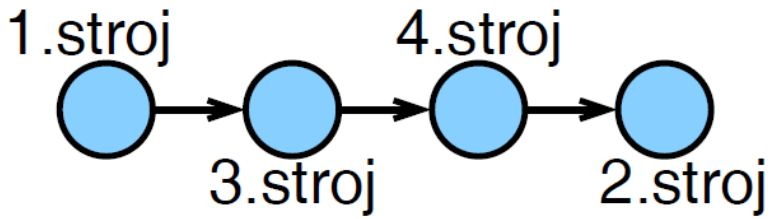
- $P3|prec|C_{max}$: bike assembly
- $Pm|r_j|\sum w_j C_j$: parallel machines

Machine Environment α [Rud13, Nie10]

- **Single machine ($\alpha = 1$)**: $1|\ldots|\ldots$
- **Identical parallel machines** *Pm*
  - m identical machines working in parallel with the same speed
  - each job consist of a single operation,
  - each job processed by any of the machines m for p_j time units
- **Uniform parallel machines** *Qm*
  - processing time of job j on machine i propotional to its speed v_i
  - $p_{ij} = p_j / v_i$
  - ex. several computers with processor different speed
- **Unrelated parallel machines** *Rm*
  - machine have different speed for different jobs
  - machine i process job j with speed v_{ij}
  - $p_{ij} = p_j / v_{ij}$
  - ex. vector computer computes vector tasks faster than a classical PC

Shop Problems [Rud13, Nie10]

- **Shop Problems**
  - each tasks is executed sequentially on several machine
    - job j consists of several operations (i, j)
    - operation (i, j) of jobs j is performed on machine i withing time p_{ij}
    - ex: job j with 4 operations $(1, j), (2, j), (3, j), (4, j)$



- Shop problems are classical
  studied in details in **operations research**
- Real problems are ofter more complicated
  - utilization of knowledge on subproblems or simplified problems in solutions

Flow shop α [Rud13, Nie10]

- **Flow shop** *Fm*
  - m machines in series
  - each job has to be processed on each machine
  - all jobs follow the same route:
    - first machine 1, then machine 2, ...
  - if the jobs have to be processed in the same order on all machines, we have a **permutation** flow shop
- **Flexible flow shop** *FFs*
  - a generalizatin of flow shop problem
  - s phases, a set of parallel machine is assigned to each phase
  - i.e. flow shop with s parallel machines
  - each job has to be processed by all phase in the same order
    - first on a machine of phase 1, then on a machine of phase 2, ...
  - the task can be performed on any machine assigned to a given phase

Open shop & job shop [Rud13, Nie10]

- **Job shop** *Jm*
  - flow shop with m machines
  - each job has its individual predetermined route to follow
    - processing time of a given jobs might be zero for some machines
  - $(i,j) \rightarrow (k,j)$ specifies that job j is performed on machine i earlier than on machine k
  - example: $(2,j) \rightarrow (1,j) \rightarrow (3,j) \rightarrow (4,j)$
- **Open shop** *Om*
  - flow shop with m machines
  - processing time of a given jobs might be zero for some machines
  - no routing restrictions (this is a scheduling decision)

Constraints β [Rud13, Nie10]

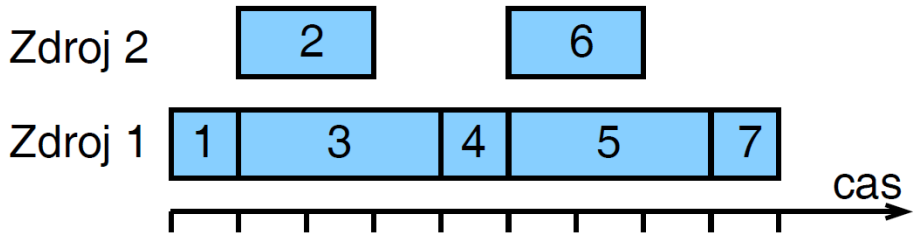
- **Precedence constraints** *prec*
  - linear sequence, tree structure
  - for jobs a, b we write $a \rightarrow b$, with meaning of $S_a + p_a \leq S_b$
  - example: bike assembly
- **Preemptions** *pmtn*
  - a job with a higher priority interrupts the current job
- **Machine suitability** M_j
  - a subset of machines M_j, on which job j can be executed
  - room assignment: appropriate size of the classroom
  - games: a computer with a HW graphical library
- **Work force constraints** W, W_l
  - another sort of machines is introduced to the problem
  - machines need to be served by operators and jobs can be performed only if operators are available, operators W
  - different groups of operators with a specific qualification can exist, W_l is a number of operators in group l

Optimization: throughput and makespan γ [Rud13]

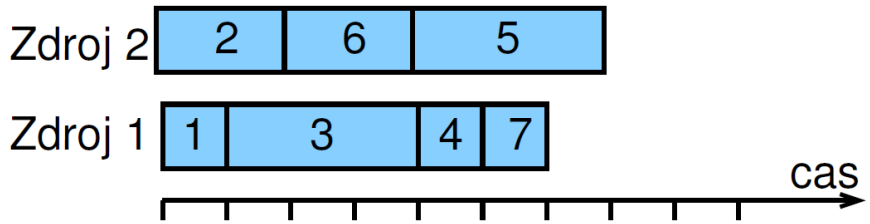
- **Makespan** C_{max}: maximum completion time

$$C_{max} = max(C_1, \ldots, C_n)$$

- Example: $C_{max} = max\{1, 3, 4, 5, 8, 7, 9\} = 9$



- Goal: **makespan minimization** often
  - maximizes **throughput**
  - ensures **uniform load of machines** (*load balancing*)
  - example: $C_{max} = max\{1, 2, 4, 5, 7, 4, 6\} = 7$



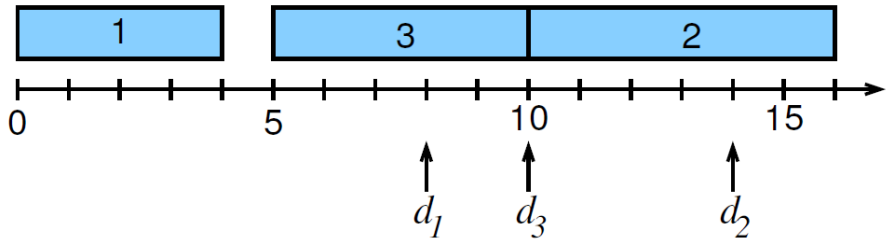
  - It is a basic criterion that is used very often.

Optimization: Lateness γ [Rud13]

- **Lateness** of job j: $L_{max} = C_j - d_j$
- **Maximum lateness** L_{max}

$$L_{max} = max(L_1, \ldots, L_n)$$

- Goal: **maximum lateness minimization**
- Example:

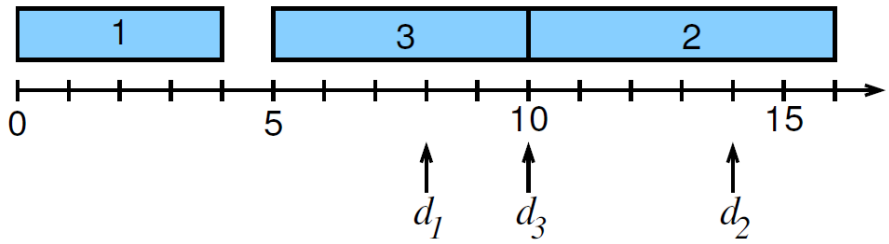


$$\begin{aligned} L_{max} &= max(L_1, L_2, L_3) = \\ &= max(C_1 - d_1, C_2 - d_2, C_3 - d_3) = \\ &= max(4 - 8, 16 - 14, 10 - 10) = \\ &= max(-4, 2, 0) = 2 \end{aligned}$$

Optimization: tardiness γ [Rud13]

- **Tardiness** úlohy j: $T_j = max(C_j - d_j, 0)$
- **Total tardiness**

$$\sum_{j=1}^{n} T_j$$



- Goal: **total tardiness minimization**
- Example: $T_1 + T_2 + T_3 =$

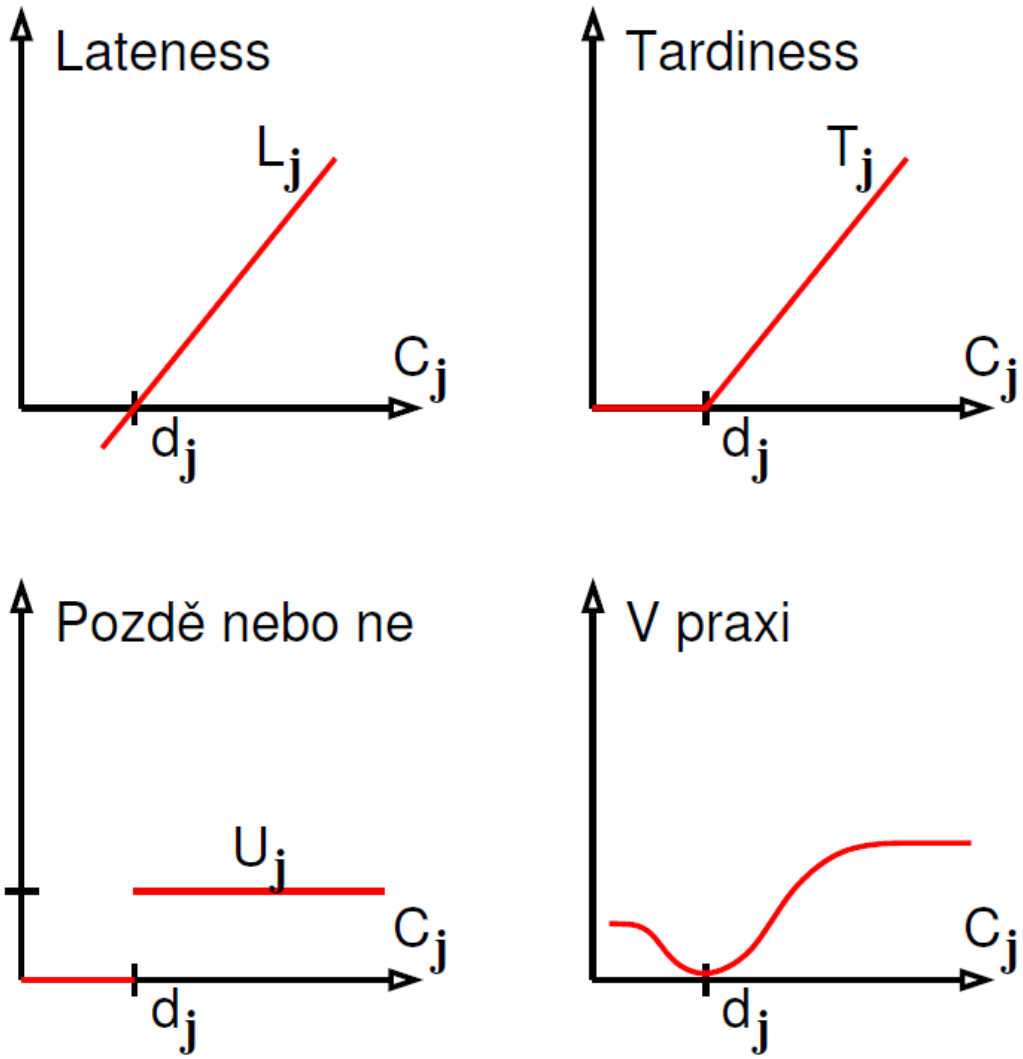
$$\begin{aligned} &= \max(C_1 - d_1, 0) + \max(C_2 - d_2, 0) + \max(C_3 - d_3, 0) = \\ &= \max(4 - 8, 0) + \max(16 - 14, 0) + \max(10 - 10, 0) = \\ &= 0 + 2 + 0 = 2 \end{aligned}$$

- **Total weighted tardiness**

$$\sum_{j=1}^{n} w_j T_j$$

- Goal: **total weighted tardiness minimization**

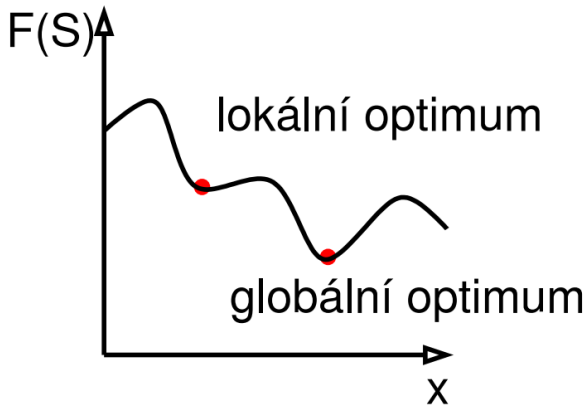
Criteria Comparison γ [Rud13]



Constructive vs. local methods [Rud13]

- **Constructive methods**
  - Start with the empty schedule
  - Add step by step other jobs to the schedule so that the schedule remains consistent
- **Local search**
  - Start with a complete non-consistent schedule
    - trivial: random generated
  - Try to find a better "similar" schedule by local modifications.
  - Schedule quality is evaluated using optimization criteria
    - ex. makespan
  - optimization criteria assess also schedule consistency
    - ex. a number of vialoted precedence constraints
  - **Hybrid approaches**
    - combinations of both methods

Local Search Algorithm [Rud13]



1. **Initialization**
   - $k = 0$
   - Select an initial schedule S_0
   - Record the current best schedule: $S_{best} = S_0$ a $cost_{best} = F(S_0)$
2. **Select and update**
   - Select a schedule from neighborhood: $S_{k+1} \in N(S_k)$
   - if no element $N(S_k)$ satisfies schedule acceptance criterion then the algorithms finishes
   - if $F(S_{k+1}) < cost_{best}$ then $S_{best} = S_{k+1}$ a $cost_{best} = F(S_{k+1})$
3. **Finish**
   - if the stop constraints are satisfied then the algorithms finishes
   - otherwise $k = k + 1$ and continue with step 2.

Single machine + nonpreemptive jobs [Rud13]

- **Schedule representation**
  - permutations n jobs
  - example with six jobs: $1, 4, 2, 6, 3, 5$
- **Neighborhood definition**
  - pairwise exchange of neighboring jobs
    - $n - 1$ possible schedules in the neighborhood
    - example: $1, 4, 2, 6, 3, 5$ is modified to $1, 4, 2, 6, 5, 3$
  - or select an arbitrary job from the schedule and place it to an arbitrary position
    - $\leq n(n-1)$ possible schedules in the neighborhood
    - example: from $1, 4, 2, 6, 3, 5$ we select randomly 4 and place it somewhere else: $1, 2, 6, 3, 4, 5$

Criteria for Schedule Selection [Rud13]

- Criteria for schedule selection
  - **Criterion for schedule acceptance/refuse**
- The main difference among a majority of methods
  - to accept a better schedule all the time?
  - to accept even worse schedule sometimes?
- methods
  - probabilistic
    - **random walk**: with a small probability (ex. 0.01) a worse schedule is accepted
    - **simulated annealing**
  - deterministic
    - **tabu search**: a tabu list of several last state/modifications that are not allowed for the following selection is maintained

Tabu Search [Rud13]

- **Deterministic criterion for schedule acceptance/refuse**
- **Tabu list** of several last schedule modifications is maintained
  - each new modification is stored on the top of the tabu list
    - ex. of a store modification: exchange of jobs j and k
  - tabu list = a list of forbidden modifications
  - the neighborhood is constrained over schedules, that do not require a change in the tabu list
    - a protection against cycling
    - example of a trivial cycling:
      the first step: exchange jobs 3 and 4, the second step: exchange jobs 4 and 3
  - a fixed length of the list (often: 5-9)
    - the oldest modifications of the tabu list are removed
    - too small length: cycling risk increases
    - too high length: search can be too constrained
- **Aspiration criterion**
  - determines when it is possible to make changes in the tabu list
  - ex. a change in the tabu list is allowed if $F(S_{best})$ is improved.

Tabu Search Algorithm [Rud13]

1.
   - $k = 1$
   - Select an initial schedule S_1 using a heuristics,
     $S_{best} = S_1$
2.
   - Choose $S_c \in N(S_k)$
   - If the modification $S_k \to S_c$ is forbidden because it is in the tabu list
     then continue with step 2
3.
   - If the modification $S_k \to S_c$ is not forbidden by the tabu list
     then $S_{k+1} = S_c$,
     store the reverse change to the top of the tabu list
     move other positions in the tabu list one position lower
     remove the last item of the tabu list
   - if $F(S_c) < F(S_{best})$ then $S_{best} = S_c$
4.
   - $k = k + 1$
   - if a stopping condition is satisfied then finish
     otherwise continue with step 2.

Example: tabu list [Rud13]

A schedule problem with $1|d_j|\sum w_j T_j$

- remind: $T_j = \max(C_j - d_j, 0)$

jobs	1	2	3	4
p_j	10	10	13	4
d_j	4	2	1	12
w_j	14	12	1	12

- Neighborhood: all schedule obtained by pair exchange of neighbor jobs
- Schedule selection from the neighborhood: select the best schedule
- Tabu list: pairs of jobs (j, k) that were exchanged in the last two modifications
- Apply tabu search for the initial solution $(2, 1, 4, 3)$
- Perform four iterations

Example: tabu list - solution I [Rud13]

jobs	1	2	3	4
p_j	10	10	13	4
d_j	4	2	1	12
w_j	14	12	1	12

$S_1 = (2, 1, 4, 3)$
$F(S_1) = \sum w_j T_j = 12 \cdot 8 + 14 \cdot 16 + 12 \cdot 12 + 1 \cdot 36 = 500 = F(S_{best})$
$F(1, 2, 4, 3) = 480$
$F(2, \underline{4}, \underline{1}, 3) = 436 = F(S_{best})$
$F(2, 1, 3, 4) = 652$
Tabu list: $\{(1, 4)\}$

$S_2 = (2, 4, 1, 3), F(S_2) = 436$
$F(\underline{4}, \underline{2}, 1, 3) = 460$
$F(2, 1, 4, 3)(= 500)$ tabu!
$F(2, 4, 3, 1) = 608$
Tabu list: $\{(2, 4), (1, 4)\}$

$S_3 = (4, 2, 1, 3), F(S_3) = 460$
$F(2, 4, 1, 3)(= 436)$ tabu!
$F(4, \underline{1}, \underline{2}, 3) = 440$
$F(4, 2, 3, 1) = 632$
Tabu list: $\{(2, 1), (2, 4)\}$

Example: tabu list - solution II [Rud13]

úlohy	1	2	3	4
p_j	10	10	13	4
d_j	4	2	1	12
w_j	14	12	1	12

$S_3 = (4, 2, 1, 3), F(S_3) = 460$
$F(2, 4, 1, 3)(= 436)$ tabu!
$F(4, \underline{1}, \underline{2}, 3) = 440$
$F(4, 2, 3, 1) = 632$
Tabu list: $\{(2, 1), (2, 4)\}$

$S_4 = (4, 1, 2, 3), F(S_4) = 440$
$F(\underline{1}, \underline{4}, 2, 3) = 408 = F(S_{best})$
$F(4, 2, 1, 3)(= 460)$ tabu!
$F(4, 1, 3, 2) = 586$
Tabu list: $\{(4, 1), (2, 1)\}$

$F(S_{best}) = 408$

Summary

- **What is AI planning**
  - reaching a goal or a task problem solution using a sequence of action changing the environment,
  - classical planning as a search for a sequence of actions in state space that transforms the initial state to a goal state.
- **Representations**
  - STRIPS specifies modifications of the world through changes of satisfied closed atoms,
  - PDDL defines a language format that enable to record STRIPS planning domain and STRIPS planning problem
- **Planning Methods**
  - creation of a plan as a searching method through world state space
- **Scheduling**
  - methodology
  - tabu search

7 Příloha

- PDDL Specification

PDDL Domains [Wic11]

<domain> ::=
(define (domain <name>)
[<extension-def>]
[<require-def>]
[<types-def>]:typing
[<constants-def>]
[<domain-vars-def>]:expression-evaluation
[<predicates-def>]
[<timeless-def>]
[<safety-def>]:safety-constraints
<structure-def>*)

<extension-def> ::=
(:extends <domain name>+)
<require-def> ::=
(:requirements <require-key>+)
<require-key> ::=
:strips | :typing | ...

<types-def> ::= (:types <typed list (name)>)
<constants-def> ::=
(:constants <typed list (name)>)
<domain-vars-def> ::=
(:domain-variables
<typed list(domain-var-declaration)>)
<predicates-def> ::=
(:predicates <atomic formula skeleton>+)
<atomic formula skeleton> ::=
(<predicate> <typed list (variable)>)
<predicate> ::= <name>
<variable> ::= ?<name>
<timeless-def> ::=
(:timeless <literal (name)>+)
<structure-def> ::= <action-def>
<structure-def> ::=:domain-axioms <axiom-def>
<structure-def> ::=:action-expansions <method-def>

PDDL Types [Wic11]

- PDDL types syntax

  <typed list (x)> ::= x*

  <typed list (x)> ::=:typing

  x+ - <type> <typed list(x)>

  <type> ::= <name>

  <type> ::= (either <type>+)

  <type> ::=:fluents (fluent <type>)

PDDL Example: DWR Types [Wic11]

```
(define (domain dock-worker-robot)

    (:requirements :strips :typing )

    (:types
        location      ;there are several connected locations
        pile          ;is attached to a location,
                      ;it holds a pallet and a stack of containers
        robot         ;holds at most 1 container,
                      ;only 1 robot per location
        crane         ;belongs to a location to pickup containers
        container )

...)
```

PDDL Example: Predicates [Wic11]

```
(:predicates
    (adjacent ?l1 ?l2 - location)          ;location ?l1 is adjacent to ?l2
    (attached ?p - pile ?l - location)     ;pile ?p attached to location ?l
    (belong ?k - crane ?l - location)      ;crane ?k belongs to location ?l

    (at ?r - robot ?l - location)          ;robot ?r is at location ?l
    (occupied ?l - location)               ;there is a robot at location ?l
    (loaded ?r - robot ?c - container )    ;robot ?r is loaded with container ?c
    (unloaded ?r - robot)                  ;robot ?r is empty

    (holding ?k - crane ?c - container)    ;crane ?k is holding a container ?c
    (empty ?k - crane)                     ;crane ?k is empty

    (in ?c - container ?p - pile)          ;container ?c is within pile ?p
    (top ?c - container ?p - pile)         ;container ?c is on top of pile ?p
    (on ?c1 - container ?c2 - container)   ;container ?c1 is on container ?c2
)
```

PDDL Action [Wic11]

<action-def> ::=
(:action <action functor>
:parameters (<typed list (variable)>)
<action-def body>)
<action functor> ::= <name>
<action-def body> ::=
[:vars (<typed list(variable)>)]:existential-preconditions :conditional-effects
[:precondition <GD>]
[:expansion <action spec>]:action−expansions
[:expansion :methods]:action−expansions
[:maintain <GD>]:action−expansions
[:effect <effect>]
[:only-in-expansions <boolean>]:action−expansions

PDDL Goal Specification [Wic11]

<GD> ::= <atomic formula(term)>
<GD> ::= (and <GD>+)
<GD> ::= <literal(term)>
<GD> ::=:disjunctive-preconditions (or <GD>+)
<GD> ::=:disjunctive-preconditions (not <GD>)
<GD> ::=:disjunctive-preconditions (imply <GD> <GD>)
<GD> ::=:existential-preconditions (exists (<typed list(variable)>) <GD>)
<GD> ::=:universal-preconditions (forall (<typed list(variable)>) <GD>)
<literal(t)> ::= <atomic formula(t)>
<literal(t)> ::= (not <atomic formula(t)>)
<atomic formula(t)> ::= (<predicate> t*)
<term> ::= <name>

PDDL Effects [Wic11]

<effect> ::= (and <effect>+)

<effect> ::= <atomic formula(term)>

<effect> ::= (not <atomic formula(term)>)

<effect> ::=:conditional-effects
(forall (<variable>*) <effect>)

<effect> ::=:conditional-effects
(when <GD> <effect>)

<effect> ::=:fluents (change <fluent> <expression>)

PDDL Example: Operator [Wic11]

```
;; moves a robot between two adjacent locations
(:action move
   :parameters (?r - robot ?from ?to - location)
   :precondition (and
      (adjacent ?from ?to) (at ?r ?from)
      (not (occupied ?to)))
   :effect (and
      (at ?r ?to) (occupied ?to)
      (not (occupied ?from)) (not (at ?r ?from)) ))
```

PDDL Problem [Wic11]

<problem> ::= (define (problem <name>)
(:domain <name>)
[<require-def>]
[<situation>]
[<object declaration>]
[<init>]
<goal>+
[<length-spec>])
<object declaration> ::= (:objects <typed list (name)>)
<situation> ::= (:situation <initsit name>)
<initsit name> ::= <name>
<init> ::= (:init <literal(name)>+)
<goal> ::= (:goal <GD>)
<goal> ::=:action−expansions (:expansion <action spec(action-term)>)
<length-spec> ::= (:length [(:serial <integer>)] [(:parallel <integer>)])

PDDL Problem: DWR example [Wic11]

```
;; a simple DWR problem with 1 robot and 2
locations
(define (problem dwrpb1)
      (:domain dock-worker-robot)
      (:objects
            r1 - robot
            l1 l2 - location
            k1 k2 - crane
            p1 q1 p2 q2 - pile
            ca cb cc cd ce cf pallet - container)
      (:init
            (adjacent l1 l2)
            (adjacent l2 l1)
            (attached p1 l1)
            (attached q1 l1)
            (attached p2 l2)
            (attached q2 l2)
            (belong k1 l1)
            (belong k2 l2)

            (in ca p1) (in cb p1) (in cc p1)
            (on ca pallet) (on cb ca) (on cc cb)
            (top cc p1)

            (in cd q1) (in ce q1) (in cf q1)
            (on cd pallet) (on ce cd) (on cf ce)
            (top cf q1)

            (top pallet p2)
            (top pallet q2)

            (at r1 l1)
            (unloaded r1)
            (occupied l1)

            (empty k1)
            (empty k2))

      ;; task is to move all containers to locations l2
      ;; ca and cc in pile p2, the rest in q2
      (:goal (and
            (in ca p2) (in cc p2)
            (in cb q2) (in cd q2) (in ce q2) (in cf q2))))
```

Literatura I

Dana Nau.

CMSC 722, ai planning (fall 2009), lecture notes.
http://www.cs.umd.edu/class/fall2009/cmsc722/, 2009.

Tim Nieberg.

Lecture corse "scheduling".
http://www.or.uni-bonn.de/lectures/ss10/sched10.html, July 2010.

Stuart J. Russell and Peter Norvig.

Artificial Intelligence, A Modern Approach.
Pre, third edition, 2010.

Hana Rudová.

PA167 Rozvrhování, lecture notes, in Czech.
http://www.fi.muni.cz/ hanka/rozvrhovani/, March 2013.

Gerhard Wickler.

A4m33pah, lecture notes.
http://cw.felk.cvut.cz/doku.php/courses/a4m33pah/prednasky, February 2011.