Search and Heuristics for Classical Planning

Approaches to solve Classical planning problems

- **Explicit search**
- **SAT planning**
- **Hierarchical Task Networks**

There are some others, but we'll stick to those

Satisficing or Optimal?

must carefully distinguish:

- satisficing planning: any plan is OK (cheaper ones preferred)
- optimal planning: plans must have minimum cost

solved by similar techniques, but:

- details very different
- almost no overlap between best techniques for satisficing planning and best techniques for optimal planning
- many tasks that are trivial for satisficing planners are impossibly hard for optimal planners

Explicit Search

Overall Idea

- Apply standard search algorithms to planning problem:
  - search space is subset of state space
  - nodes correspond to world states (search states)
  - arcs correspond to state transitions
  - path in the search space corresponds to plan

Classes of search

- 2 classes of search methods:
  - Uninformed (blind)
    - The search has no information about the current state, but is only able to tell if it's goal or not. Explores everything
  - Informed
    - The search has some way for telling the "quality" of the current state, and for informing the selection of the next state to visit.

Examples of uninformed search

- Breadth-first search
- Depth-first search

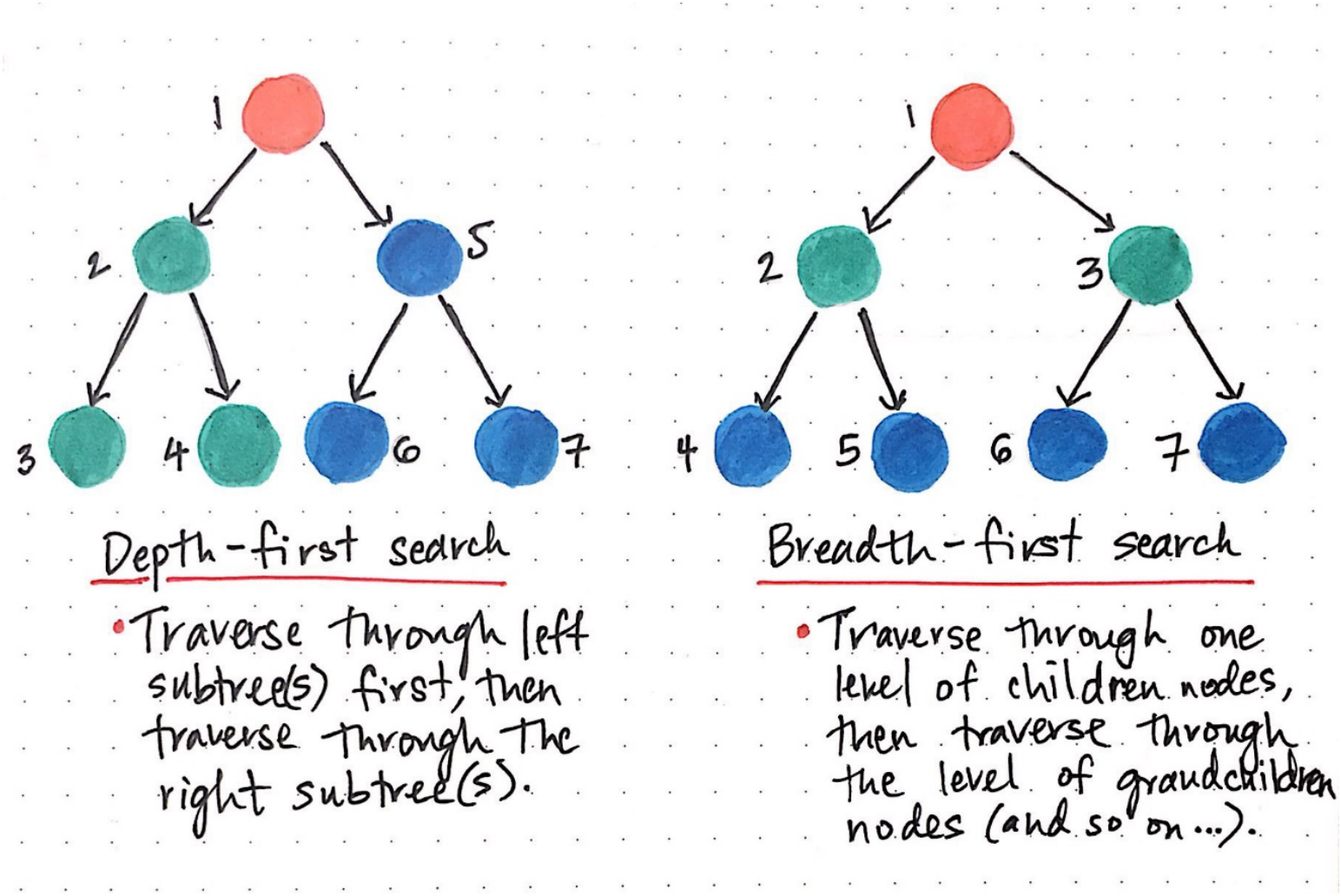


Image taken from: https://bit.ly/2DrNVO5

What is a heuristic?

A heuristic value is an estimation of the remaining cost for reaching the goal.

It is used for deciding which among several alternative courses of action promises to be the most effective in order to achieve the goal

It is basically a sort of compass: you are in a state, and you need to decide the best way to go to reach your goal.

Informed

- Systematic:
  - Greedy Best First (GBF)
  - A*, wA*, IDA*, etc.
- Local search
  - Hill climbing, simulated annealing, etc.

A large number of search approaches is available, we are only mentioning a few here

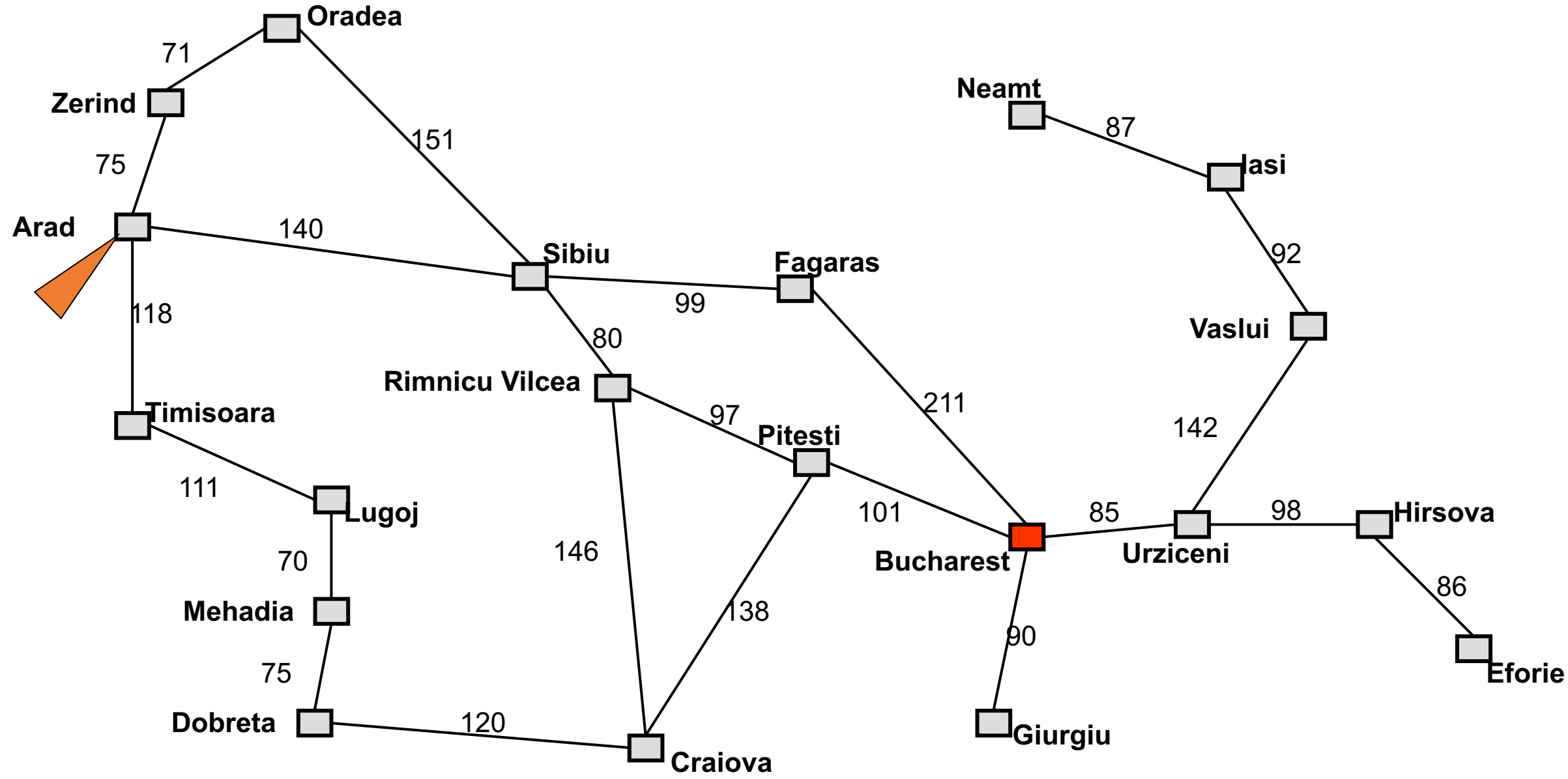
GBF

- use heuristic function as evaluation function:

$$f(n) = h(n)$$

  - always expands the node that is closest to the goal node
  - eats the largest chunk out of the remaining distance, hence, “greedy”

A traditional problem: tour of Romania

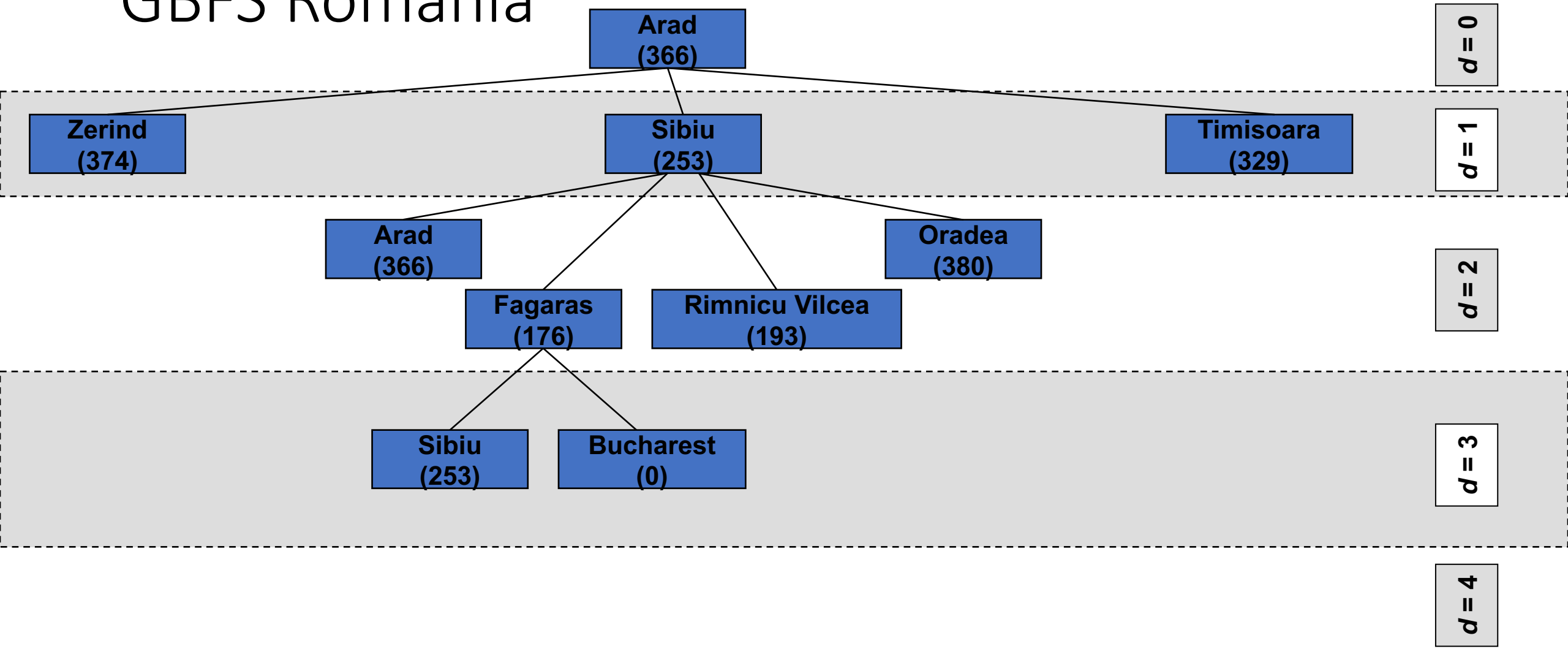


Heuristic

- $h_{SLD}(n)$ = straight-line distance to Bucharest

Arad	366	Hirsova	151	Rimnicu Vilcea	193
Bucharest	0	Iasi	226		
Craiova	160	Lugoj	244	Sibiu	253
Dobreta	242	Mehadia	241	Timisoara	329
Eforie	161	Neamt	234	Urziceni	80
Fagaras	176	Oradea	380	Vaslui	199
Giurgiu	77	Pitesti	100	Zerind	374

GBFS Romania



A* search

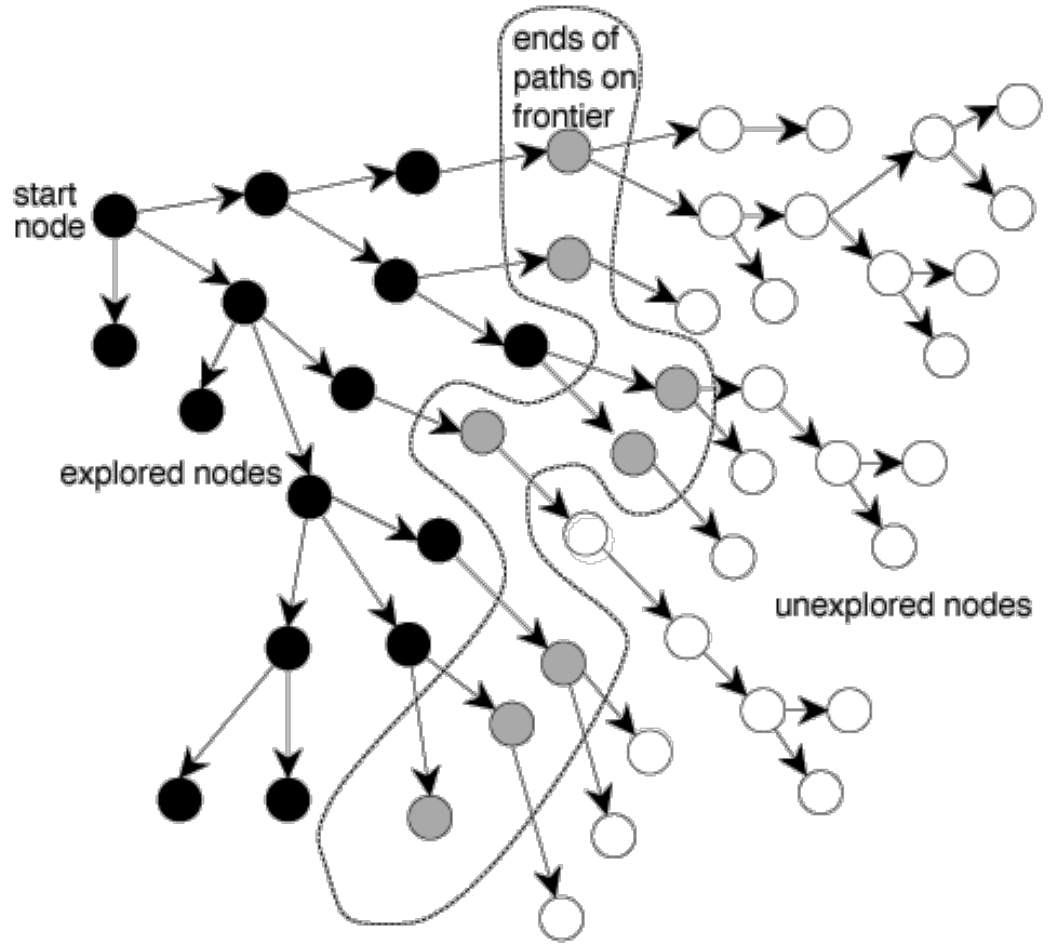
- best-first search where

$$\boldsymbol{f(n) = h(n) + g(n)}$$

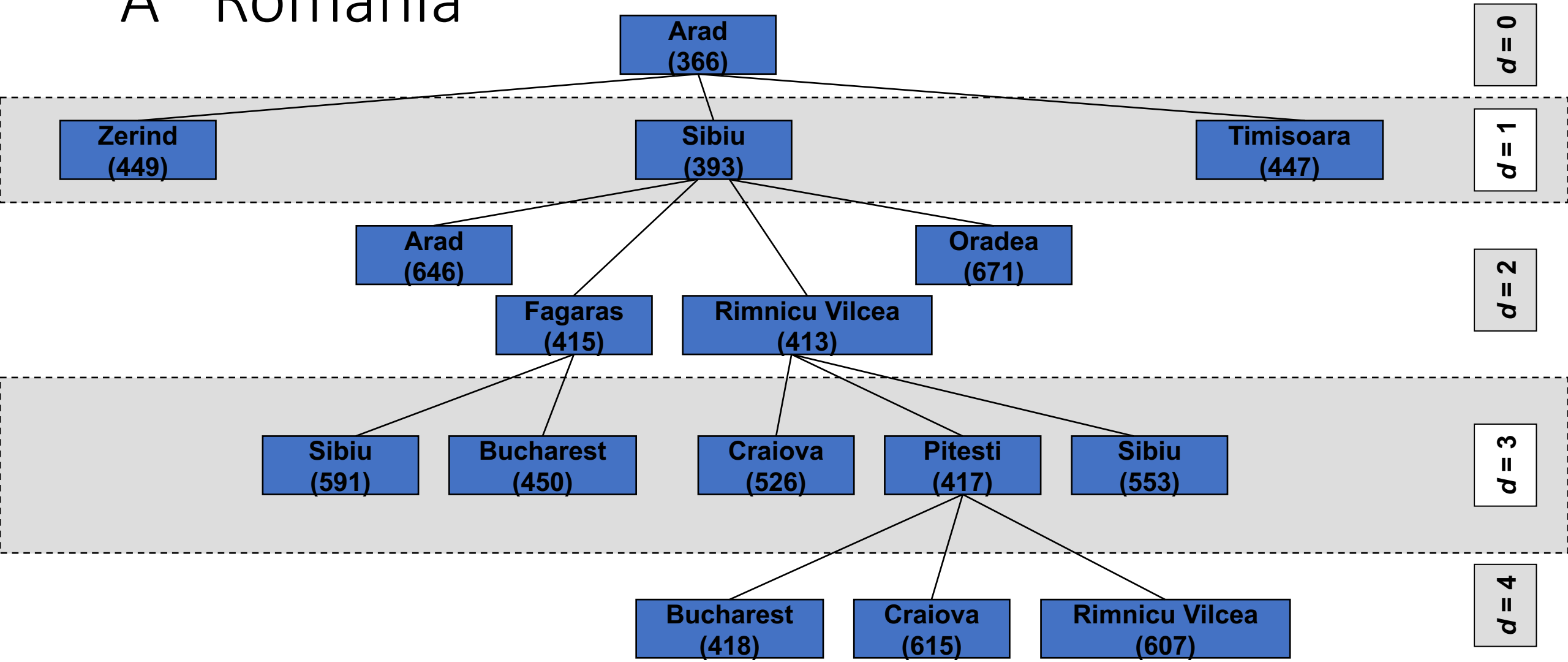
  - $h(n)$ the heuristic function (as before)
  - $g(n)$ the cost to reach the node n
- evaluation function:

  $f(n)$ = estimated cost of the cheapest solution through n

High level view



A* Romania


Arad
(366)
Zerind
(449)
Sibiu
(393)
Timisoara
(447)
Arad
(646)
Oradea
(671)
Fagaras
(415)
Rimnicu Vilcea
(413)
Sibiu
(591)
Bucharest
(450)
Craiova
(526)
Pitesti
(417)
Sibiu
(553)
Bucharest
(418)
Craiova
(615)
Rimnicu Vilcea
(607)
d = 0
d = 1
d = 2
d = 3
d = 4

wA*-

As A*, but:

$$f(n) = w*h(n) + g(n)$$

The weight w ∈ R + 0 is an algorithm parameter:

- For W = 0, weighted A ∗ behaves like?
- For W = 1, weighted A ∗ behaves like?
- For W = 100000, weighted A ∗ behaves like?

A different dimension

- So fare we looked at forward search
  - Search forward from initial state to goal.
  - Search states = world states.
- Backward
  - Search backward from goal to initial state.
  - Search states = sub-goals we would need to achieve.

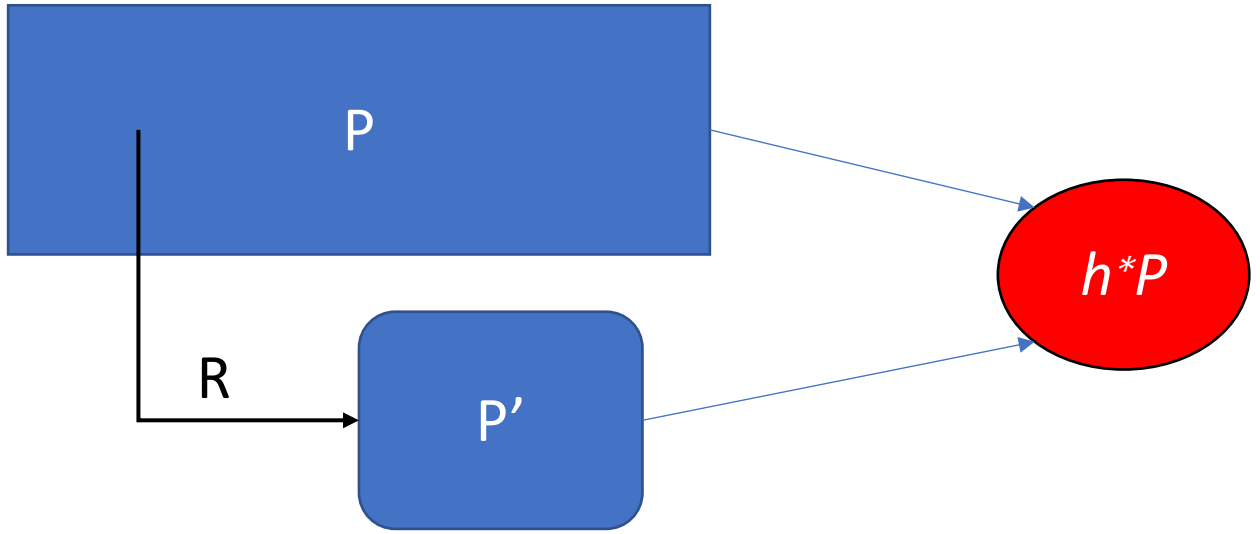
Local search?

- You start from a state and:
  - try to iteratively improve it by applying operators
  - by taking random steps,
  - or by restarting

- Works well when problems are very complex and hard to address in other ways, do not give guarantees on solutions.

Something more on Heuristics

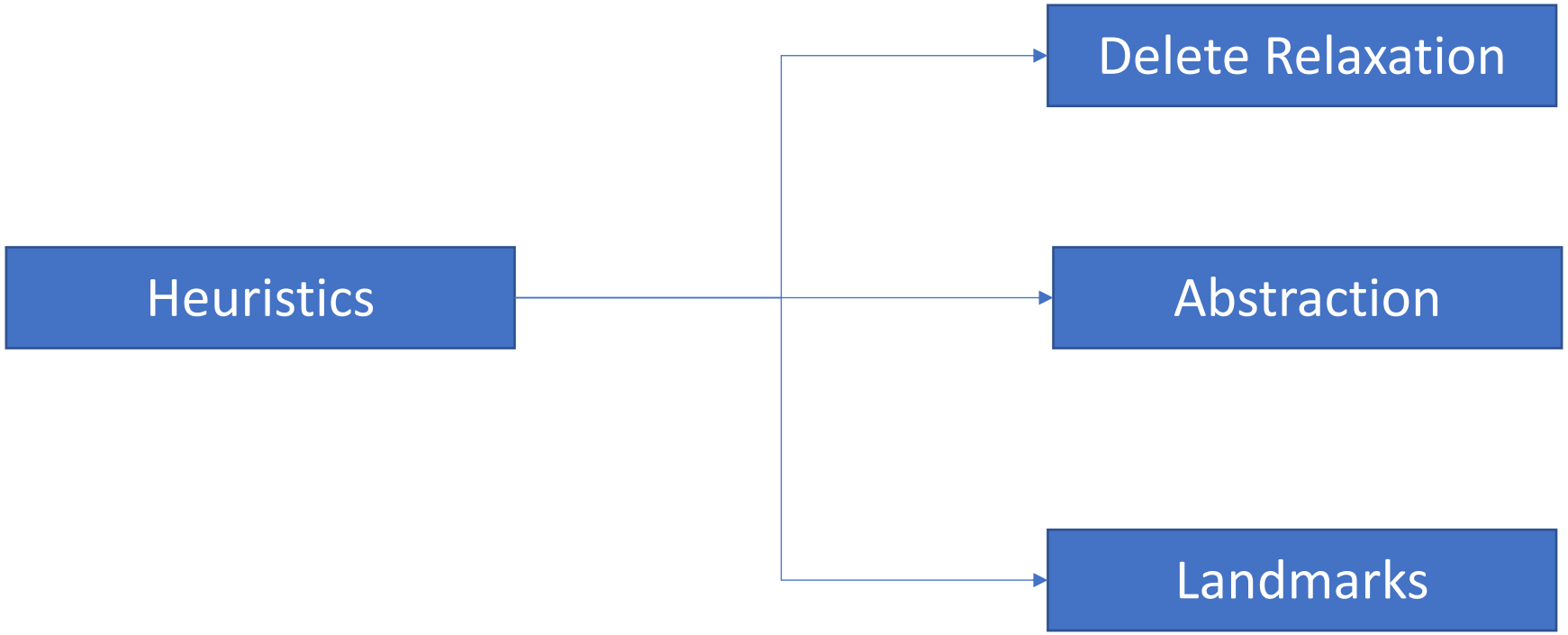
Heuristic?



- You have a class *P* of problems, whose perfect cost h^*P you wish to estimate.
- You define a class *P’* of simpler problems, whose perfect cost is cheaper to compute and can be used to estimate h^*P.
- You define a transformation – the relaxation mapping **R** – that maps instances of *P* into instances of *P’*.

Image courtesy of A. Torralba and C. Croitoru

(some) Classes of heuristics



Focus only on classical (deterministic) Planning

Delete relaxation

- Delete relaxation is very wide-spread, and highly successful for satisficing planning!

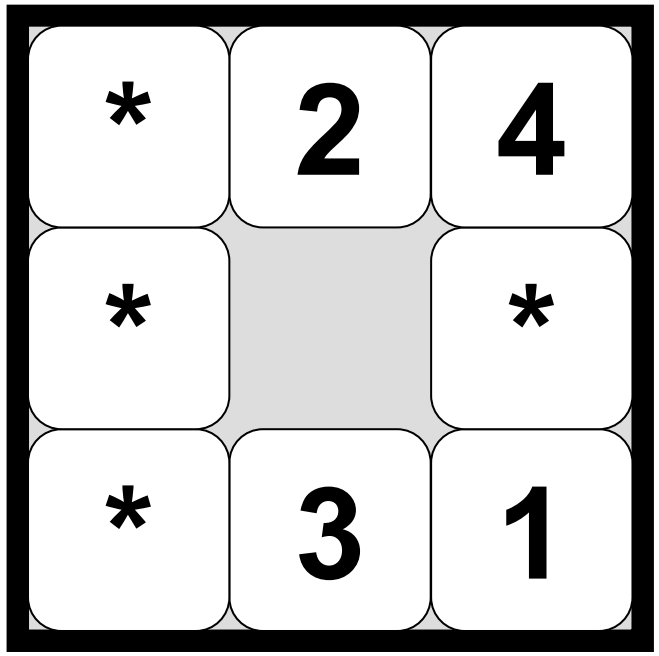
- Super easy idea: ignore negative effects of planning actions. Solve the corresponding relaxed problem:
  - the size of the relaxed plan is the heuristic value.

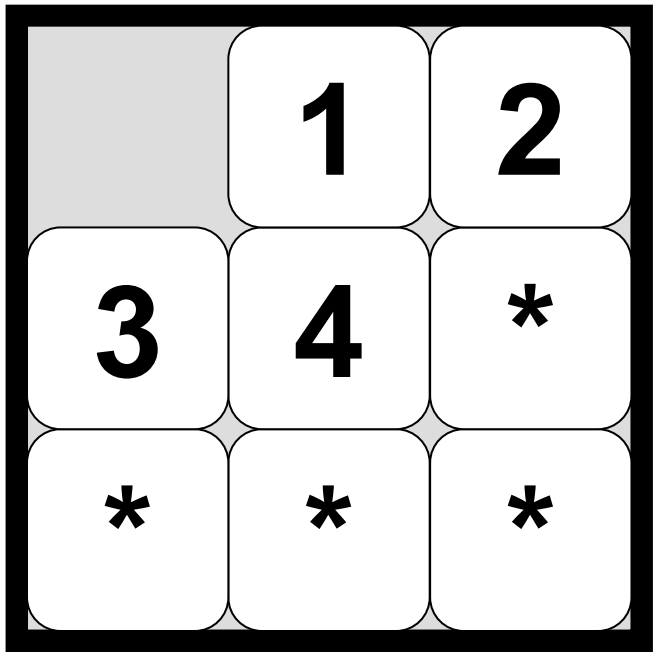
Abstraction

- Idea: reducing complexity by abstracting
- Abstracting means dropping some distinctions between states, while preserving the overall transition behaviour as much as possible.
- There are numerous techniques that deal with finding a good domain-independent way for abstracting
  - Two main classes: pattern databases and merge-and-shrink

Pattern databases

- Pattern database (PDB) heuristics are one of the two most commonly used classes of abstraction heuristics in planning

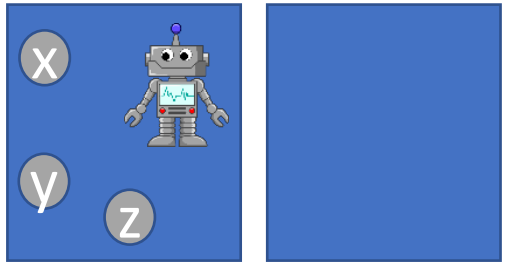




"Abstract the planning task by choosing a subset P of variables (the pattern), and ignoring the values of all other variables."

Landmarks

- Basic Idea: A landmark is something that every plan for the task must satisfy at some point



- Can be like:
  - some operator must be applied
  - some atom must be true
  - some formula must be true

- In our example, we know that the robot will need to be in room B at some point.

Landmarks

- How to use them during search as heuristics?
  - Find landmarks in a pre-process to planning.
  - Heuristic value(state) := number of yet unachieved landmarks.
    - (“Number of open items on the to-do list”)

- For our example, if we consider as landmark the fact that the robot has to be in room B, the initial heuristic value is 1
  - The fact that the robot has to hold the balls is not a landmark, because it can either hold with gripper R or L.

Merging Heuristics

- Usually, some heuristics work well only on some specific kind of problems (structures)
  - There is the need for either selecting or combining different heuristics
- Ways for combining:
  - Sum, max, etc.
  - Cost partitioning (mostly used for optimal planning)

Cost Partitioning

- Cost partitioning distributes the operator costs of a task among multiple abstractions in such a way that the sum of abstraction heuristics is guaranteed to be admissible
- In other words, it allows to combine different PDBs in a principled way.
  - The optimal cost partitioning is hard to compute, but there are approximate ways for doing it.
  - More in the references at end.

SAT Planning

Basic idea: reformulate the problem

- formalise problem of finding plan with a given horizon (length bound) as a propositional satisfiability problem and feed it to a generic SAT solver
- to obtain a (semi-) complete algorithm, try with increasing horizons until a plan is found(= the formula is satisfiable)
- important optimization: allow applying several non-conflicting actions "at the same time" so that a shorter horizon suffices

Similar approaches work for ASP, for instance, or other formalisms

SAT encoding: variables

$$\neg a \wedge (b \vee \neg c \vee d) \wedge (a \vee \neg d)$$

given a propositional planning task $\langle V,I,O,\gamma \rangle$

given horizon $T \in N0$

Propositional variables v_i for all $v \in V$, $0 \leq I \leq T$ encode
state after i steps of the plan

Propositional variables o_i for all $o \in O$, $1 \leq I \leq T$ encode operator(s)
applied in i-th step of the plan

Design choices

- SAT encoding
  - Sequential or parallel
  - Most focus on SAT planning given to this aspect
- SAT solver
  - Off-the-shelf solvers such as MiniSAT or Glucose
  - Planning-specific solvers
- Evaluation strategy
  - Advance horizon T +1, or more aggressively
  - Check one horizon at the time or more

Planning Engine example

Madagascar

- Websites:
  - https://research.ics.aalto.fi/software/sat/madagascar/
  - https://users.aalto.fi/~rintanj1/satplan.html

- Based on aggressive evaluation strategy
  - More horizons checked in parallel

- Excellent performance in IPC 2014

Hierarchical Task Networks

Motivation

- We may already have an idea how to go about solving problems in a planning domain
- Example: travel to a destination that's far away:
  - Domain-independent planner:
    - many combinations of vehicles and routes
  - Experienced human:
    - small number of "recipes" e.g., flying:
      - 1. buy a ticket from local airport to remote airport
      - 2. travel to local airport
      - 3. fly to remote airport
      - 4. travel to final destination
- How to allow planning systems to exploit such recipes?

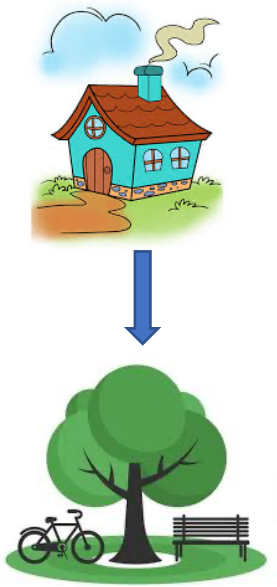
HTN Planning

- Allows to describe actions and subtasks that do fit the recipe.
- Ingredients:
  - States
  - Tasks
  - Operators
  - Methods
  - planning algorithm

HTN in a Nutshell

- HTN planning is done by **problem reduction**: the planner recursively decomposes tasks into subtasks, stopping when it reaches *primitive* tasks that can be performed directly by *planning operators*.
- In order to tell the planner how to decompose non-primitive tasks into subtasks, it needs to have a set of *methods*, where each method is a schema for decomposing a particular kind of task into a set of subtasks (provided that some set of preconditions is satisfied).
- For each task, there may be more than one applicable method, and thus more than one way to decompose the task into subtasks. The planner may have to try several of these alternative decompositions before finding one that is solvable at a lower level.

States and Tasks



- State: description of the current situation
  - I'm at home, I have £20, there's a park 8 km away
- Task: description of an activity to perform
  - Travel to the park
- Two kinds of tasks
  - **Primitive** task: a task that corresponds to a basic action
  - **Compound** task: a task that is composed of other simpler tasks

Operators

- Operators: parameterized descriptions of what the basic actions do
  - walk from location x to location y
    - Precond: agent is at x
    - Effects: agent is at y
  - call taxi to location x
    - Precond: (none)
    - Effects: taxi is at x
  - ride taxi from location x to location y
    - Precond: agent and taxi are at x
    - Effects: agent and taxi at y, agent owes 1.50 + ½ distance(x,y)
  - pay driver
    - Precond: agent owes amount of money r, agent has money m ≥ r
    - Effects: agent owes nothing, agent has money m – r
- Actions: operators with arguments

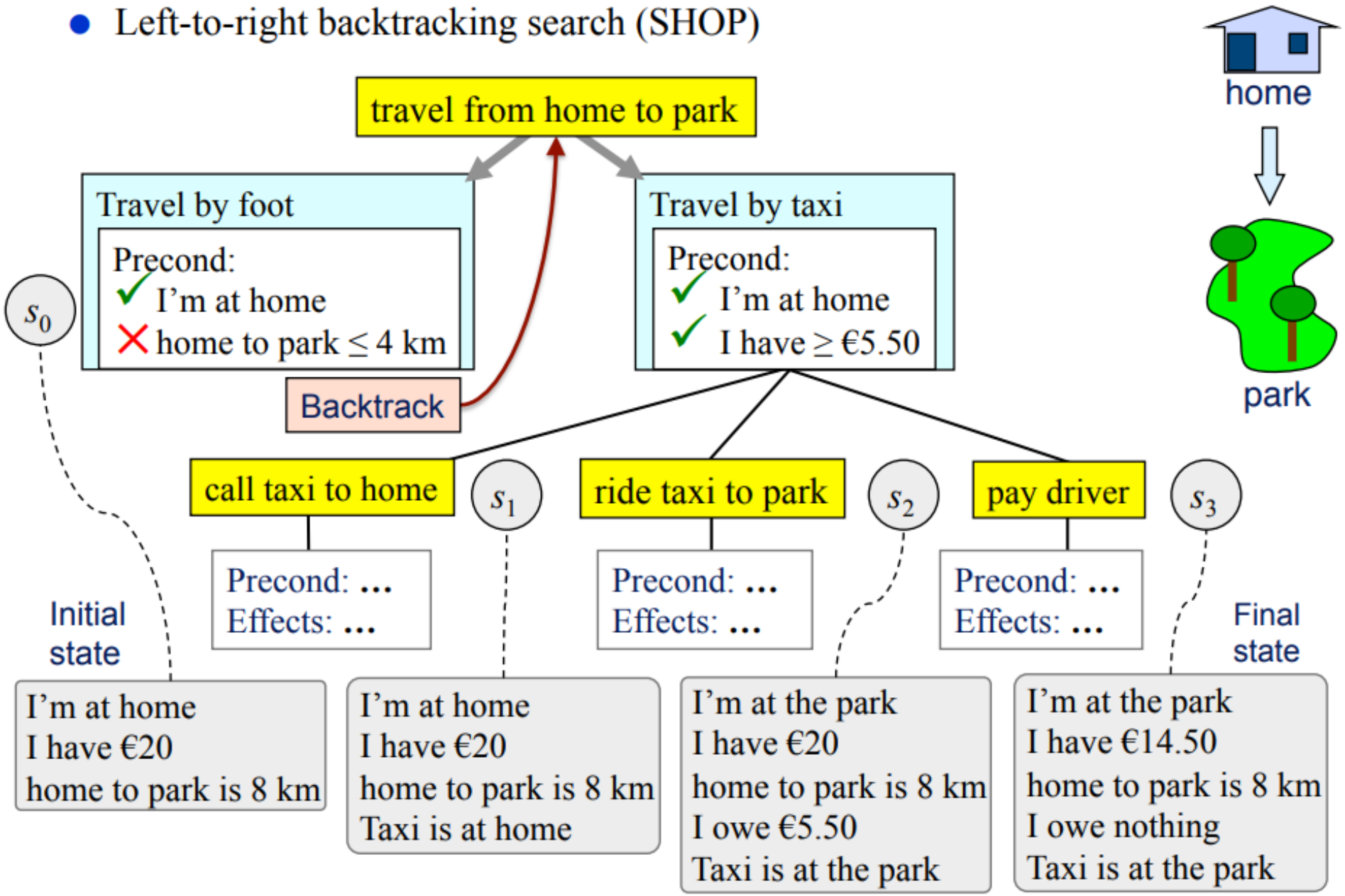
Methods

- Method: parametrised description of a possible way to perform a compound task by performing a collection of subtasks

- There may be more than one method for the same task
  - **travel by foot from x to y**
    - *Task*: travel from x to y
    - *Precond*: agent is at x, distance to y is ≤ 4 km
    - *Subtasks*: walk from x to y
  - **travel by taxi from x to y**
    - *Task*: travel from x to y
    - *Precond*: agent is at x, agent has money ≥ 1.5 + ½ distance(x,y)
    - *Subtasks*: call taxi to x, ride taxi from x to y, pay driver

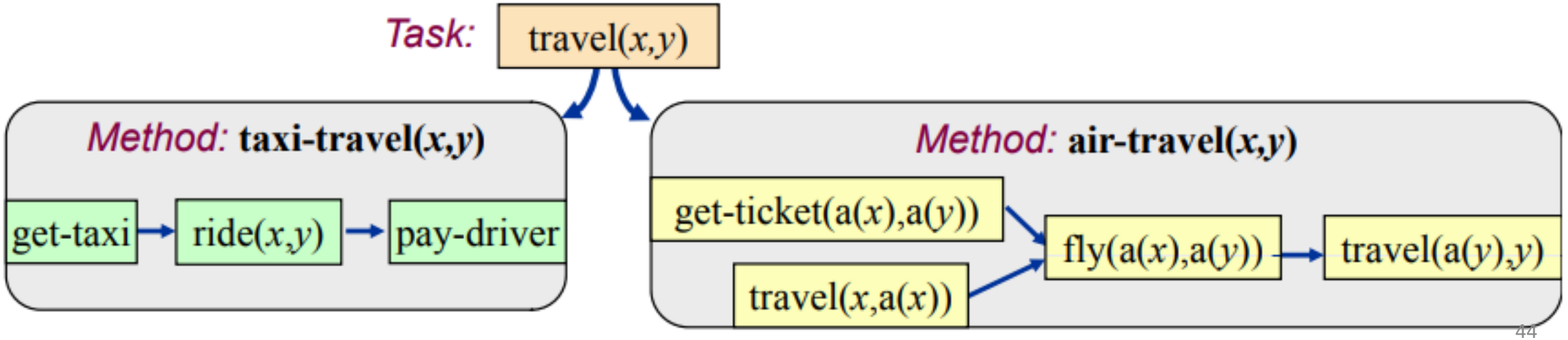
Example

- Left-to-right backtracking search (SHOP)



HTN Planners

- HTN planners may be domain-specific
  - e.g., robotics and bridge (cards game)
- Or they may be domain-configurable
  - Domain-independent planning engine
  - Domain description that defines not only the operators, but also the methods
  - Problem description
    - domain description initial state initial task network



Plan-Space Search

Part of the explicit search, but with some differences

State- vs Plan- space search

- **state-space search**:
  search through world states, generally represented using graphs or trees
- **plan-space search**:
  search through graph of partial plans
  - nodes: partially specified plans
  - arcs: plan refinement operations
  - solutions: partial-order plans

What is a partial plan?

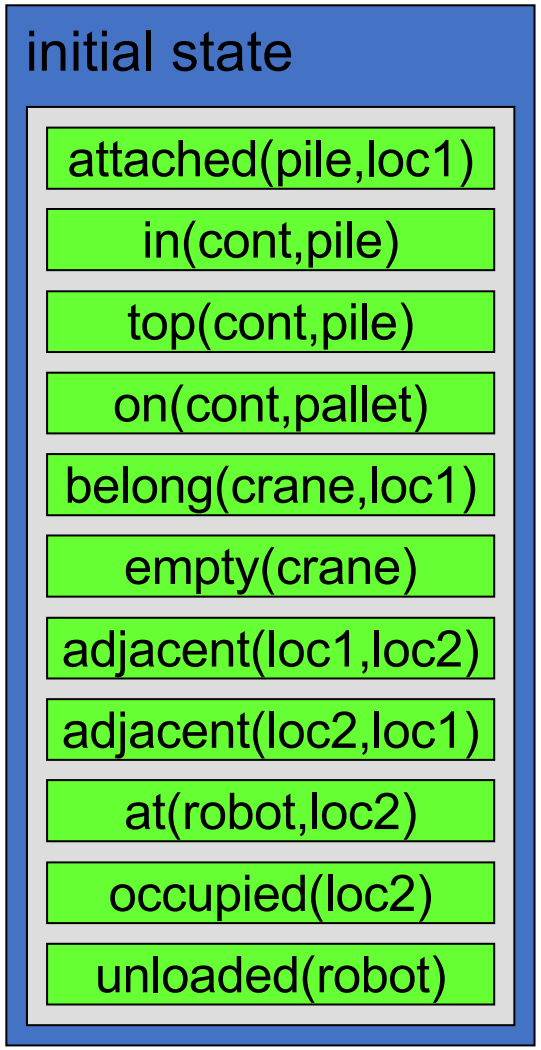
- partial plan:
  - subset of the actions
  - subset of the overall structure
    - temporal ordering of actions
    - rationale: what the action achieves in the plan
  - subset of variable bindings
  - Initial and goal states presented as actions!

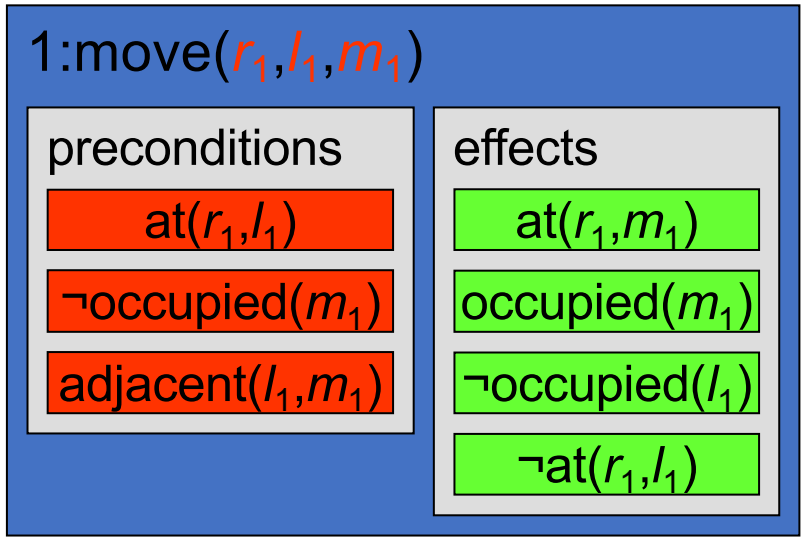
Adding actions

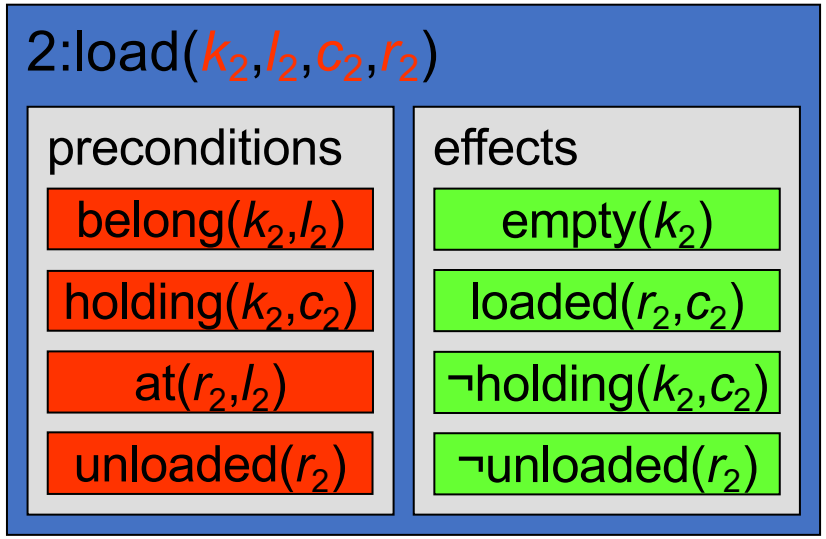
- partial plan contains actions
  - initial state
  - goal conditions
  - set of operators with different variables

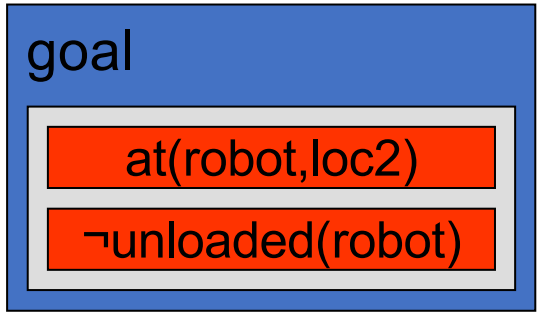
- reason for adding new actions
  - to achieve unsatisfied preconditions
  - to achieve unsatisfied goal conditions

Example


initial state
attached(pile,loc1)
in(cont,pile)
top(cont,pile)
on(cont,pallet)
belong(crane,loc1)
empty(crane)
adjacent(loc1,loc2)
adjacent(loc2,loc1)
at(robot,loc2)
occupied(loc2)
unloaded(robot)


1:move(r_1,l_1,m_1)
preconditions
at(r_1,l_1)
¬occupied(m_1)
adjacent(l_1,m_1)
effects
at(r_1,m_1)
occupied(m_1)
¬occupied(l_1)
¬at(r_1,l_1)


2:load(k_2,l_2,c_2,r_2)
preconditions
belong(k_2,l_2)
holding(k_2,c_2)
at(r_2,l_2)
unloaded(r_2)
effects
empty(k_2)
loaded(r_2,c_2)
¬holding(k_2,c_2)
¬unloaded(r_2)


goal
at(robot,loc2)
¬unloaded(robot)

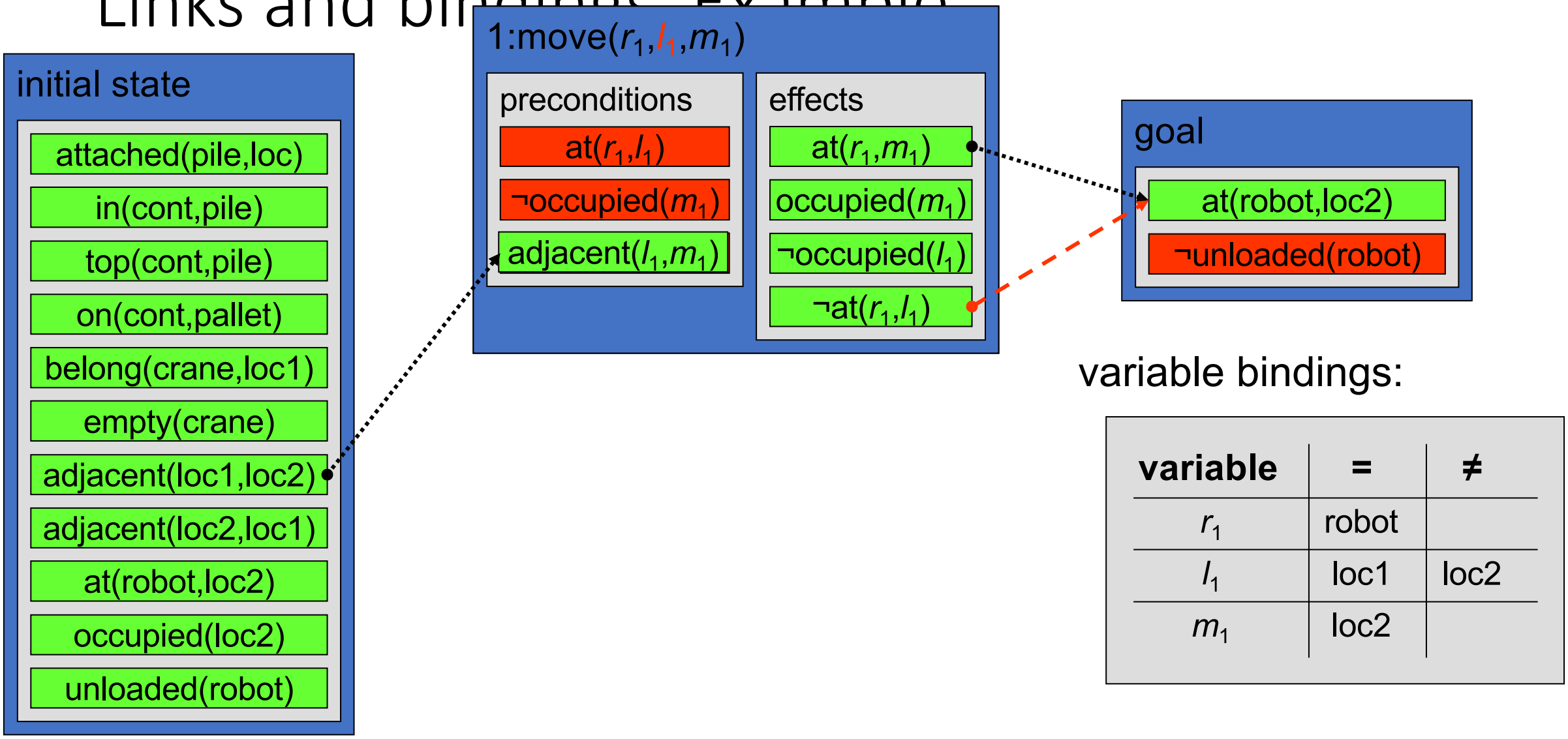
Adding links

- partial plan contains causal links
  - links from the provider
    - an effect of an action or
    - an atom that holds in the initial state
  - to the consumer
    - a precondition of an action or
    - a goal condition
- reasons for adding causal links
  - prevent interference with other actions

Variable bindings!

- partial plan contains variable bindings
  - new operators introduce new (copies of) variables into the plan
  - solution plan must contain actions
  - variable binding constraints keep track of possible values for variables and co-designation
- reasons for adding variable bindings
  - to turn operators into actions
  - to unify and effect with the precondition it supports

Links and bindings. Example



variable bindings:

variable	**=**	**≠**
r_1	robot	
l_1	loc1	loc2
m_1	loc2	

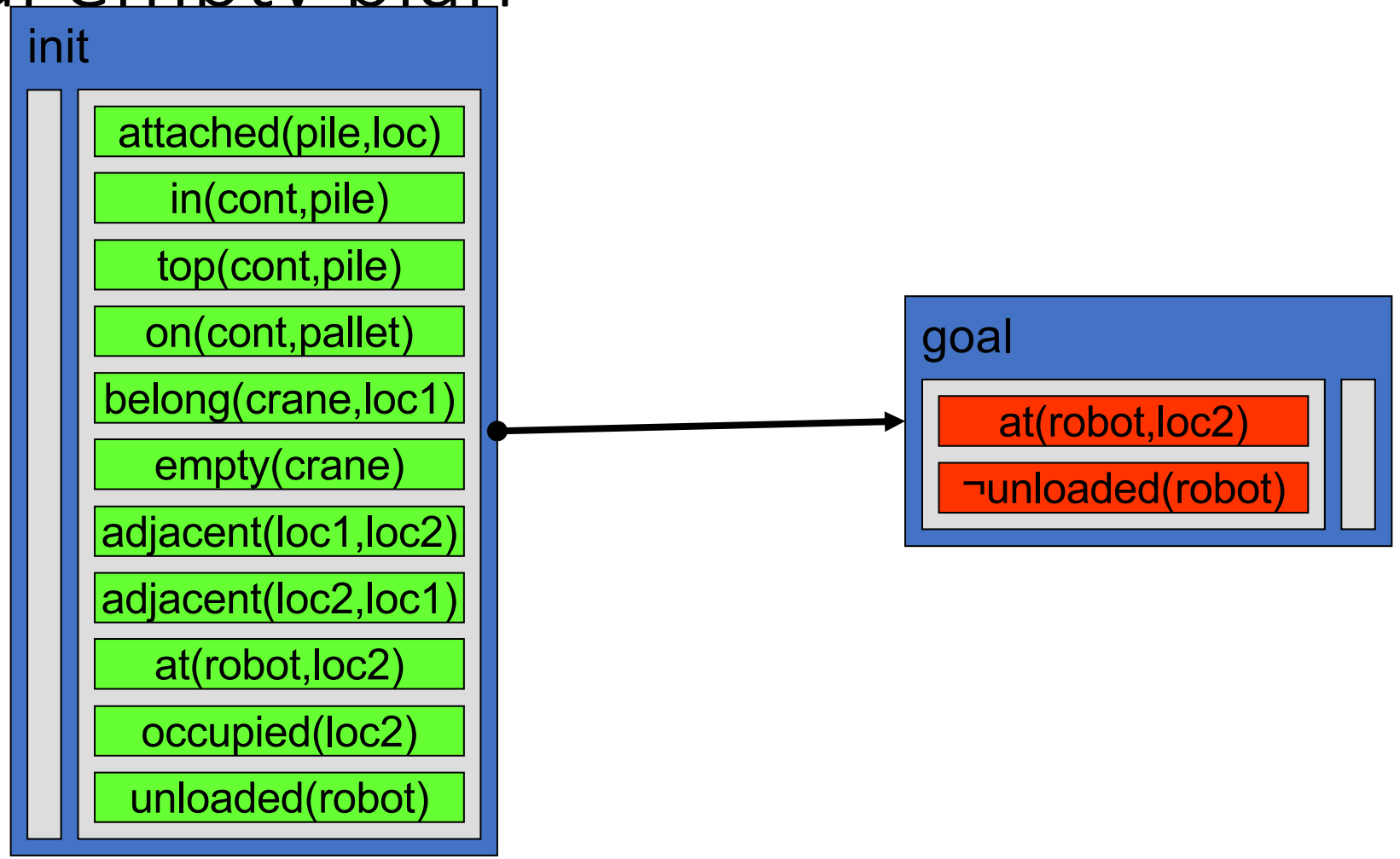
Ordering Constraints

- partial plan contains ordering constraints
  - binary relation specifying the temporal order between actions in the plan
- reasons for adding ordering constraints
  - all actions after initial state
  - all actions before goal
  - causal link implies ordering constraint
  - to avoid possible interference

Where to start from?

- initial state and goal as dummy actions
  - init: no preconditions, initial state as effects
  - goal: goal conditions as preconditions, no effects
- empty plan = ({init, goal},{(init≺goal)},{},{}):
  - two dummy actions init and goal;
  - one ordering constraint: init before goal;
  - no variable bindings; and
  - no causal links.

Initial empty plan



Summary of Plan-space elements

- Start from empty plan
- generate successor through plan refinement operators (one or more)
  - Add actions
  - Add causal links
  - Add variable bindings
  - Add ordering constraints

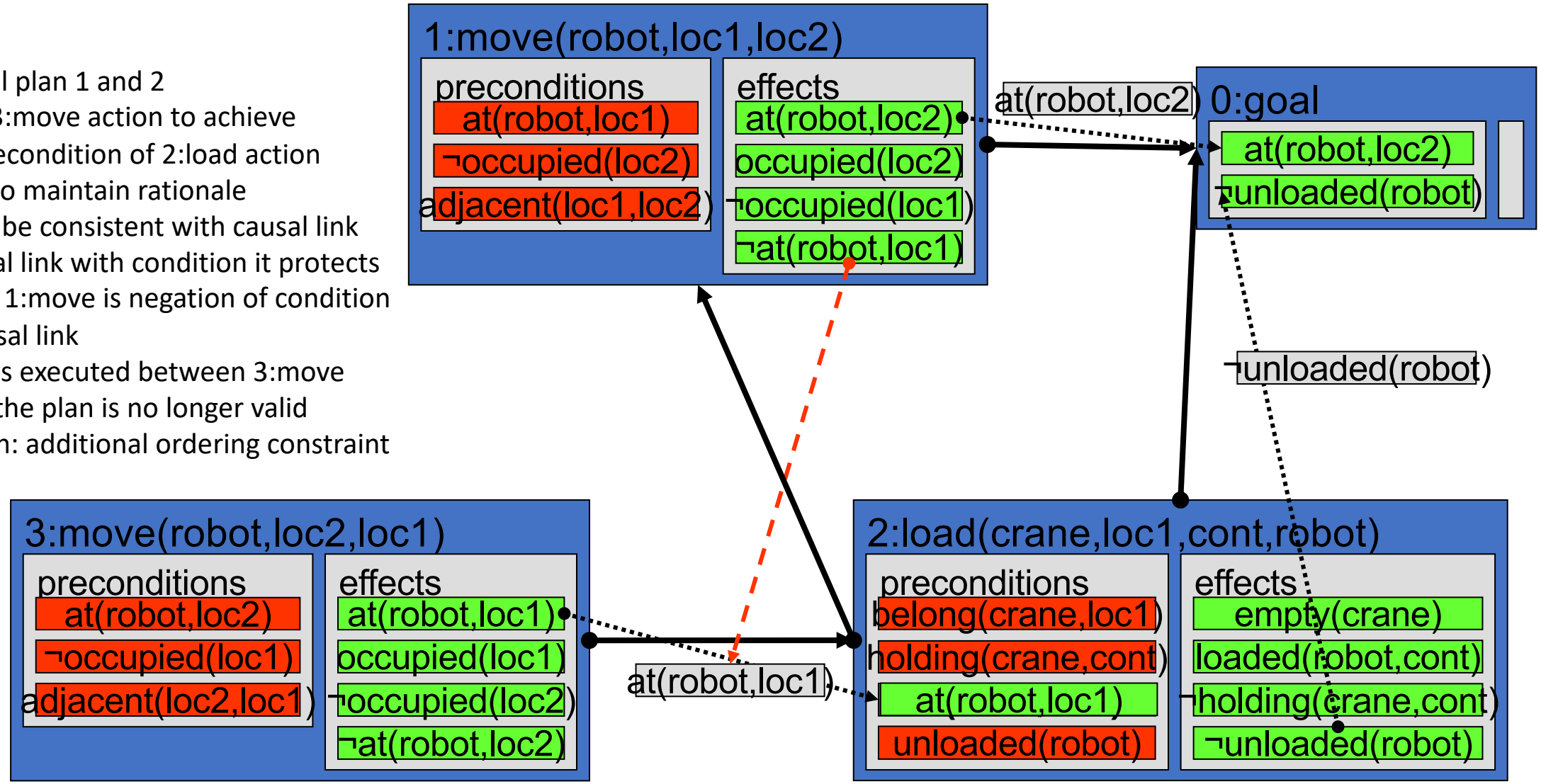
Threats

- An action a_k in a partial plan $\pi = (A, \prec, B, L)$ is a threat to a causal link $\langle a_i \text{ } -[p]\rightarrow a_j \rangle$ iff:
  - a_k has an effect $\neg q$ that is possibly inconsistent with p, i.e. q and p are unifiable;
  - the ordering constraints $(a_i \prec a_k)$ and $(a_k \prec a_j)$ are consistent with $\prec$; and
  - the binding constraints for the unification of q and p are consistent with B.

Example threat

Threat: Example

- start with partial plan 1 and 2
- introduce new 3:move action to achieve at(robot,loc1) precondition of 2:load action
- add causal link to maintain rationale
- add ordering to be consistent with causal link
- new: label causal link with condition it protects
- threat: effect of 1:move is negation of condition protected by causal link
  - if 1:move is executed between 3:move and 2:load the plan is no longer valid
- possible solution: additional ordering constraint

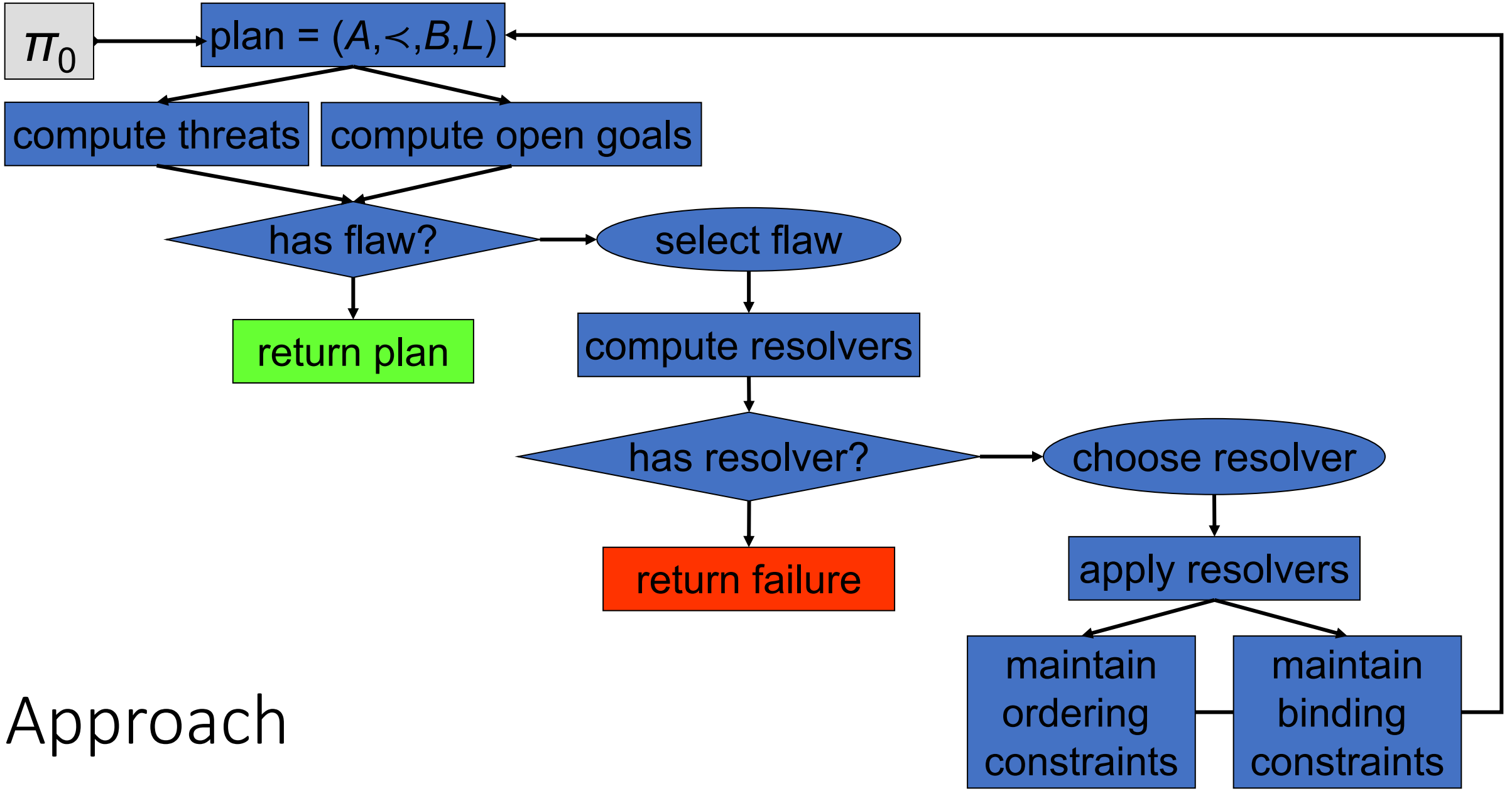


Flaws

- A flaw in a partial plan is either:
  - an unsatisfied sub-goal, i.e. a precondition of an action in *A* without a causal link that supports it; or
  - a threat, i.e. an action that may interfere with a causal link.

Finally…

- **Proposition**: A partial plan is a solution to the considered planning problem if:
  - The partial plan has no flaw;
  - the ordering constraints are not circular; and
  - the variable bindings are consistent.


π_0
plan = $(A,\prec,B,L)$
compute threats
compute open goals
has flaw?
select flaw
return plan
compute resolvers
has resolver?
choose resolver
return failure
apply resolvers
maintain ordering constraints
maintain binding constraints

Approach

References and additional infos

Pointers

- HTN
  - http://www.cs.umd.edu/~nau/papers/nau2013game.pdf
  - http://www.dia.fi.upm.es/~ocorcho/Asignaturas/ModelosRazonamiento/PresentacionesClases/planning07.pdf
- SAT
  - J. Rintanen. Engineering efficient planners with SAT, In *ECAI 2012. Proceedings of the 20th European Conference on Artificial Intelligence*, IOS Press, 2012.

Adds -- PDBs

- PDB heuristics were originally introduced for the 15-puzzle [Culberson and Schaeffer (1998)] and for Rubic's Cube [Korf (1997)].
- The first use of PDBs in planning is due to [Edelkamp (2001)]. This spawned various follow-up works [Edelkamp (2002); Haslum et al. (2005); Helmert et al. (2007); Haslum et al. (2007)].
- Manually designed PDB heuristics are currently the state of the art admissible heuristics for several search problems (e.g., 15-puzzle & Rubic's Cube).

References: PDBs

- Joseph C. Culberson and Jonathan Schaeffer. Pattern databases. Computational Intelligence, 14(3):318–334, 1998.
- Stefan Edelkamp. Planning with pattern databases. Proceedings of the 6th European Conference on Planning (ECP'01), pages 13–24. 2001.
- Stefan Edelkamp. Symbolic pattern databases in heuristic search planning. Proceedings of the 6th International Conference on Artificial Intelligence Planning and Scheduling (AIPS'02), pages 274–283, 2002.
- Stefan Edelkamp. Automated creation of pattern database search heuristics. In Proceedings of the 4th Workshop on Model Checking and Artificial Intelligence (MoChArt 2006), pages 35–50, 2006.
- Patrik Haslum, Blai Bonet, and Hector Geffner. New admissible heuristics for domain-independent planning. Proceedings of the 20th National Conference of the American Association for Artificial Intelligence (AAAI'05), pages 1163–1168, 2005.
- Patrik Haslum, Adi Botea, Malte Helmert, Blai Bonet, and Sven Koenig. Domain-independent construction of pattern database heuristics for cost-optimal planning, Proceedings of the 22nd National Conference of the American Association for Artificial Intelligence (AAAI'07), pages 1007–1012, 2007.
- Malte Helmert, Patrik Haslum, and Joerg Hoffmann. Flexible abstraction heuristics for optimal sequential planning. Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS'07), pages 176–183, 2007.
- Richard E. Korf. Finding optimal solutions to Rubik's Cube using pattern databases. Proceedings of the 14th National Conference of the American Association for Artificial Intelligence (AAAI'97), pages 700–705, 1997.

Landmarks

- The (basic) idea was proposed by the authors of LAMA [Richter et al. (2008); Richter and Westphal (2010)]. Which subsequently won two IPCs.
- LAMA uses non-admissible landmarks heuristics, basically counting the number of non-achieved fact landmarks
  - (= summing up elementary landmark heuristics induced by fact landmarks, without ensuring independence).
- One of the best admissible landmark heuristics in practice use cost partitioning [Karpas and Domshlak (2009); Helmert and Domshlak (2009)].

References: Landmarks

- Silvia Richter and Matthias Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. Journal of Artificial Intelligence Research, 39:127–177, 2010.
- Silvia Richter, Malte Helmert, and Matthias Westphal. Landmarks revisited. Proceedings of the 23rd National Conference of the American Association for Artificial Intelligence (AAAI'08), pages 975–982, 2008.
- Erez Karpas and Carmel Domshlak. Cost-optimal planning with landmarks. Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI'09), pages 1728–1733, 2009.
- Malte Helmert and Carmel Domshlak. Landmarks, critical paths and abstractions: What's the difference anyway? Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09), pages 162–169. 2009.

Cost partitioning

- Optimal Additive Composition of Abstraction-Based Admissible Heuristics. Available at:

- http://fai.cs.uni-saarland.de/katz/papers/icaps08b.pdf

- Content: Original paper proposing cost partitioning, and showing that, for certain classes of heuristics, optimal cost partitionings can be computed in polynomial time using Linear Programming. Specifically, the paper established this for abstractions as handled in this course, as well as for implicit abstractions represented through planning task fragments identified based on the causal graph.