# Machine Intelligence

## 11. Classical Planning

Automated Sequential Decision Making on "Simple" Environments

Álvaro Torralba

AALBORG UNIVERSITET

Fall 2022

Thanks to Thomas D. Nielsen and Jörg Hoffmann for slide sources

# Planning

**Ambition:**

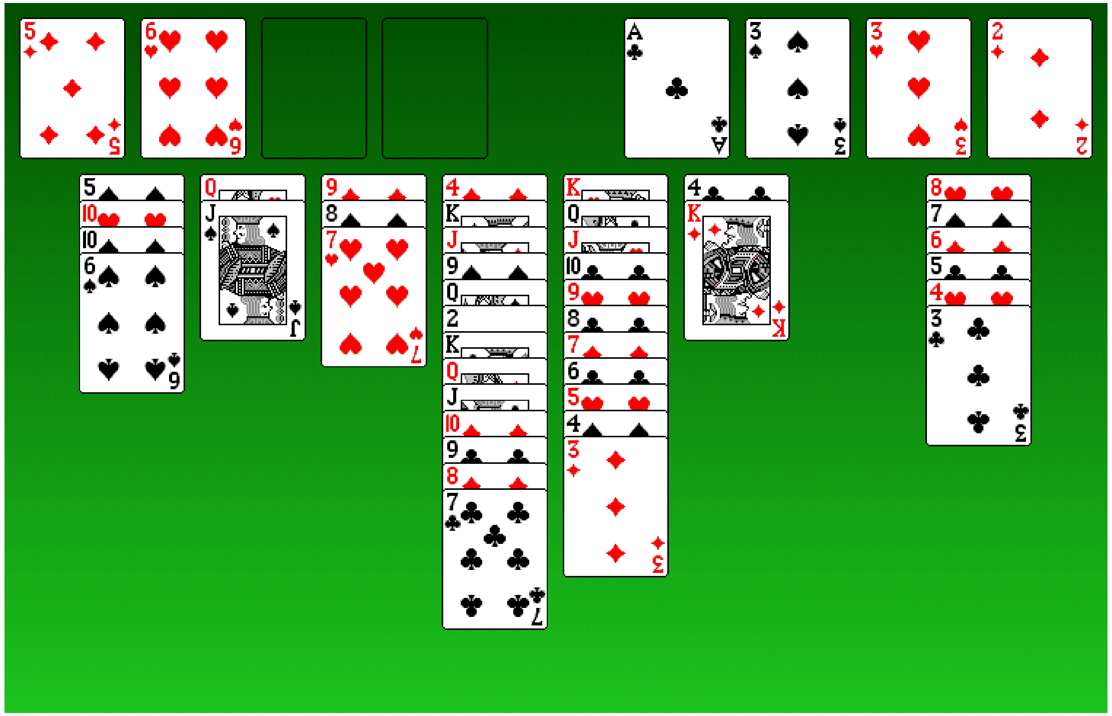
**Write one program (planner) that can solve all sequential decision-making problems.**

**How do we describe our problem to the planner?**

- A *logical description* of the possible **states**
- A *logical description* of the **initial state** $I$
- A *logical description* of the **goal condition** $G$
- *logical description* of the set $A$ of **actions** in terms of **preconditions** and **effects**

$\rightarrow$ Solution (**plan**) $=$ **sequence of actions** from $A$, transforming $I$ into a state that satisfies $G$.

# Example of a Planning Task



- **States:** Card positions (*position_Jspades=Qhearts*).
- **Actions:** Card moves (*move_Jspades_Qhearts_freecell4*).
- **Initial state:** Start configuration.
- **Goal states:** All cards "home".
- **Solution:** Card moves solving this game.

# Algorithmic Problems in Planning

**Satisficing Planning**

**Input:** A planning task $\Pi$.
**Output:** A plan for $\Pi$, or "unsolvable" if no plan for $\Pi$ exists.

**Optimal Planning**

**Input:** A planning task $\Pi$.
**Output:** An *optimal* plan for $\Pi$, or "unsolvable" if no plan for $\Pi$ exists.

→ **The techniques successful for either one of these are almost disjoint. And satisficing planning is much more effective in practice.**

→ Programs solving these problems are called (optimal) **planners**, **planning systems**, or **planning tools**.

## Our Agenda for This Chapter

- **The STRIPS Planning Formalism:** How do we represent a planning task?
  $\rightarrow$ **Lays the framework we'll be looking at.**
- **Planning Domain Definition Language:** How we actually express our classical planning problems.
  $\rightarrow$ **For reference.**
- **Planning as Heuristic Search:** How state-space search techniques are applied to solve planning tasks?
  $\rightarrow$ **A Recap of Chapter 2.**
- **The Delete Relaxation:** How to relax a planning problem?
  $\rightarrow$ **The delete relaxation is the most successful method for the automatic generation of heuristic functions. It is a key ingredient of many IPC winners during the last two decades. It relaxes STRIPS planning tasks by ignoring the delete lists.**
- **The $h^+$ Heuristic:** What is the resulting heuristic function?
  $\rightarrow$ **$h^+$ is the "ideal" delete relaxation heuristic.**
- **Approximating $h^+$:** How to actually compute a heuristic?
  $\rightarrow$ **Turns out that, in practice, we must approximate $h^+$.**

# "STRIPS" Planning

- **STRIPS** = Stanford Research Institute Problem Solver.

  *STRIPS is the simplest possible (reasonably expressive) logics-based planning language.*

- STRIPS has only **Boolean variables**: propositional logic atoms.
- Its preconditions/effects/goals are as canonical as imaginable:
  - Preconditions, goals: **conjunctions** of **positive atoms**.
  - Effects: **conjunctions** of **literals** (positive or negated atoms).
- We use the common set-based notation for this simple formalism.

→ Historical note: STRIPS [?] was originally a planner, whose language actually wasn't quite that simple.

# STRIPS Planning: Syntax

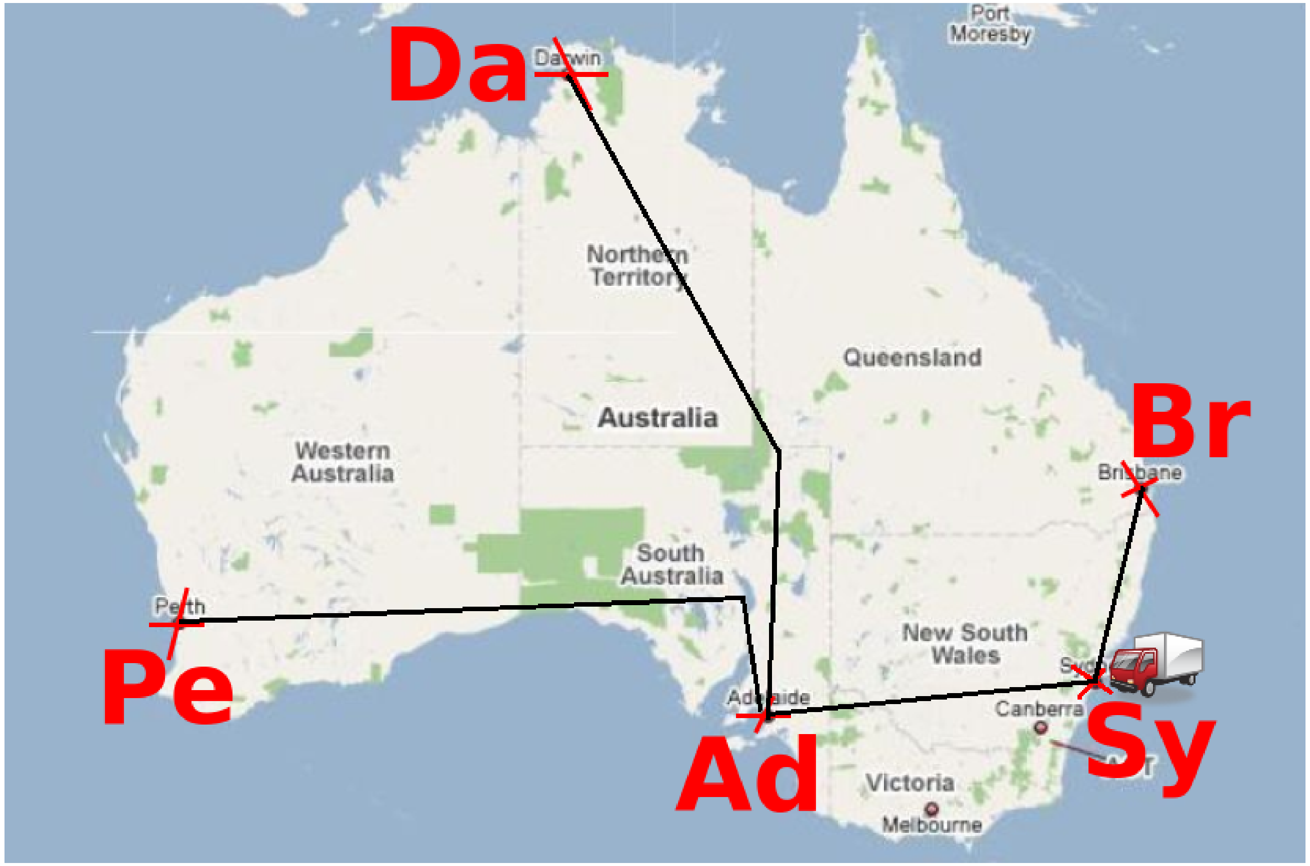
**Definition (STRIPS Planning Task).** *A* **STRIPS planning task**, *short* **planning task**, *is a 4-tuple* $\Pi = (P, A, I, G)$ *where:*

- $P$ *is a finite set of* **facts** *(aka* **propositions***).*
- $A$ *is a finite set of* **actions***; each* $a \in A$ *is a triple* $a = (pre_a, add_a, del_a)$ *of subsets of* $P$ *referred to as the action's* **precondition**, **add list**, *and* **delete list** *respectively; we require that* $add_a \cap del_a = \emptyset$*.*
- $I \subseteq P$ *is the* **initial state***.*
- $G \subseteq P$ *is the* **goal***.*

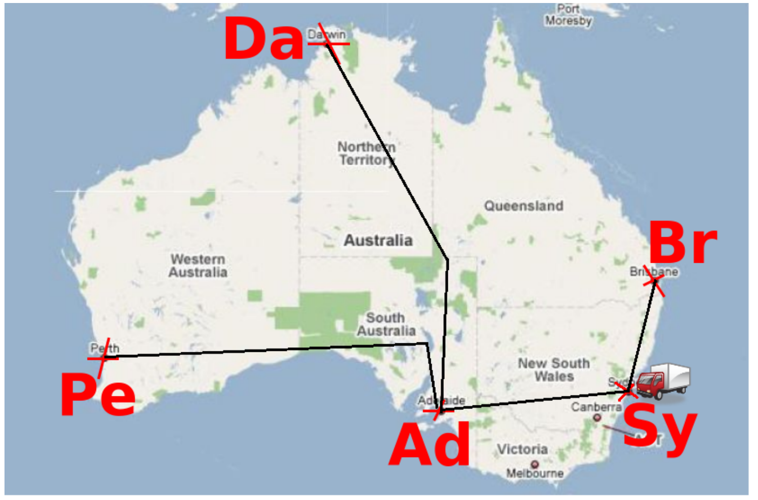
*We will often give each action* $a \in A$ *a* **name** *(a string), and identify* $a$ *with that name.*

**Note:** We assume **unit costs** for simplicity: every action has cost $1$.

# "TSP" in Australia



# STRIPS Encoding of "TSP"




- **Facts** $P$: $\{at(x), visited(x) \mid x \in \{Sydney, Adelaide, Brisbane, Perth, Darwin\}\}$.
- **Initial state** $I$:
- **Goal** $G$:
- **Actions** $a \in A$: $drive(x, y)$ where $x, y$ have a road.
  **Precondition** $pre_a$:
  **Add list** $add_a$:
  **Delete list** $del_a$:
- **Plan**:

# STRIPS Planning: Semantics

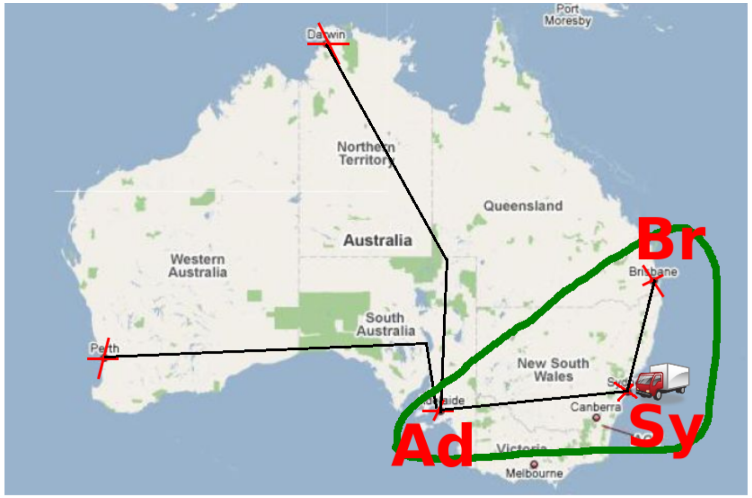
**Definition (STRIPS State Space).** *Let $\Pi = (P, A, c, I, G)$ be a STRIPS planning task. The* **state space** *of $\Pi$ is $\Theta_\Pi = (S, A, T, I, S^G)$ where:*

- *The states (also* **world states***) $S = 2^P$ are the subsets of $P$.*
- *$A$ is $\Pi$'s action set.*
- *The transitions are $T = \{s \xrightarrow{a} s' \mid pre_a \subseteq s, s' = appl(\mathbf{s}, \mathbf{a})\}$.*
  *If $pre_a \subseteq s$, then $a$ is* **applicable** *in $s$ and $appl(\mathbf{s}, \mathbf{a}) := (\mathbf{s} \cup add_\mathbf{a}) \setminus del_\mathbf{a}$. If $pre_a \not\subseteq s$, then $appl(s, a)$ is undefined.*
- *$I$ is $\Pi$'s initial state.*
- *The goal states $S^G = \{s \in S \mid G \subseteq s\}$ are those that satisfy $\Pi$'s goal.*

*An (optimal)* **plan** *for $s \in S$ is an (optimal) solution for $s$ in $\Theta_\Pi$, i.e., a path from $s$ to some $s' \in S^G$. A solution for $I$ is called a* **plan for $\Pi$**. *$\Pi$ is* **solvable** *if a plan for $\Pi$ exists.*

*For $\vec{a} = \langle a_1, \dots, a_n \rangle$, $appl(s, \vec{a}) := appl(\dots appl(appl(s, a_1), a_2) \dots, a_n)$ if each $a_i$ is applicable in the respective state; else, $appl(s, \vec{a})$ is undefined.*

# STRIPS Encoding of Simplified "TSP"

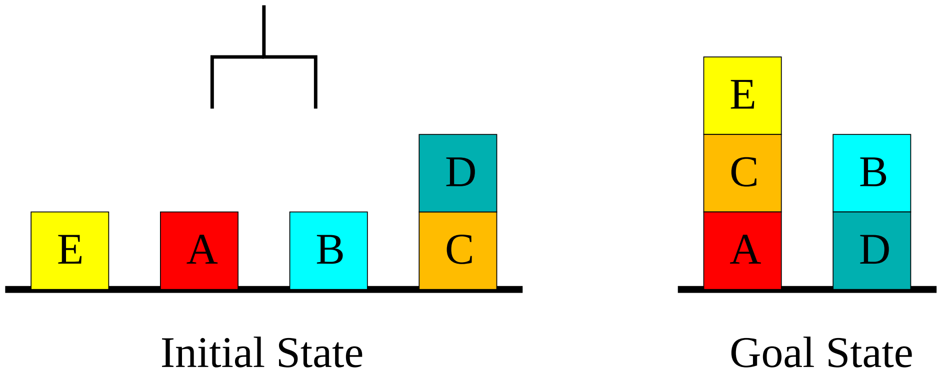



- **Facts** $P$: $\{at(x), visited(x) \mid x \in \{Sydney, Adelaide, Brisbane\}\}$.
- **Initial state** $I$:
- **Goal** $G$: $\{visited(x) \mid x \in \{Sydney, Adelaide, Brisbane\}\}$. (Note: no "$at(Sydney)$".)
- **Actions** $a \in A$: $drive(x, y)$ where $x, y$ have a road.
  - **Precondition** $pre_a$:
  - **Add list** $add_a$:
  - **Delete list** $del_a$:

# STRIPS Encoding of Simplified "TSP": State Space

$\rightarrow$ **Is this actually the state space?**

# (Oh no it's) The Blocksworld




- **Facts**: $on(x, y)$, $onTable(x)$, $clear(x)$, $holding(x)$, $armEmpty()$.
- **Initial state**: $\{onTable(E), clear(E), \dots, onTable(C), on(D, C), clear(D), armEmpty()\}$.
- **Goal**: $\{on(E, C), on(C, A), on(B, D)\}$.
- **Actions**: $stack(x, y)$, $unstack(x, y)$, $putdown(x)$, $pickup(x)$.
- $stack(x, y)$**?**

# Questionnaire

## Question!

**Which are correct encodings (part of some correct overall encoding) of the STRIPS Blocksworld $pickup(x)$ action schema?**

**(A):** $(\{onTable(x), clear(x), armEmpty()\}, \{holding(x)\}, \{onTable(x)\})$.

**(B):** $(\{onTable(x), clear(x), armEmpty()\}, \{holding(x)\}, \{armEmpty()\})$.

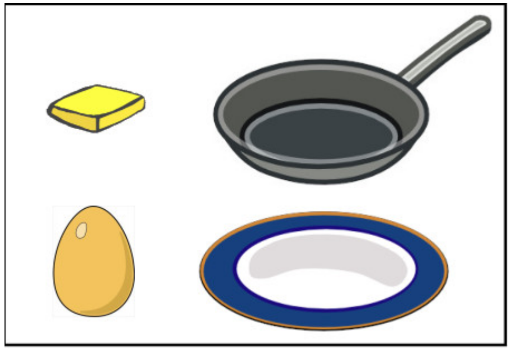
**(C):** $(\{onTable(x), clear(x), armEmpty()\}, \{holding(x)\}, \{onTable(x), armEmpty(), clear(x)\})$.

**(D):** $(\{onTable(x), clear(x), armEmpty()\}, \{holding(x)\}, \{onTable(x), armEmpty()\})$.

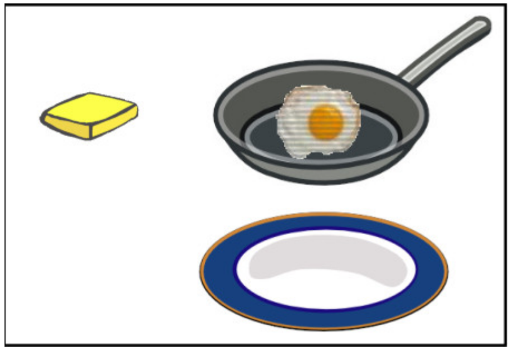
# Levels of Representation (Chapter 1)

We consider 3 representation schemes

**State based:**
State-space Search
(Chapter 2)

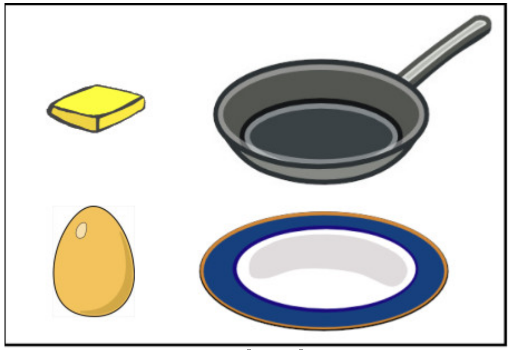


State 01

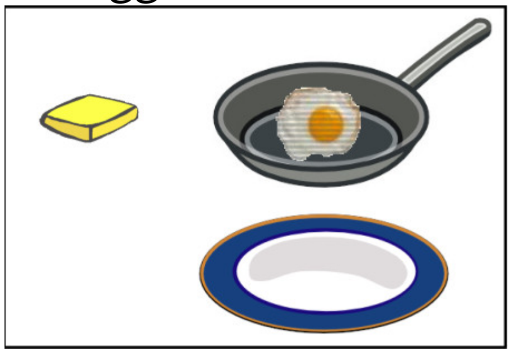


State 12

**Feature based:**
STRIPS
(This Chapter)

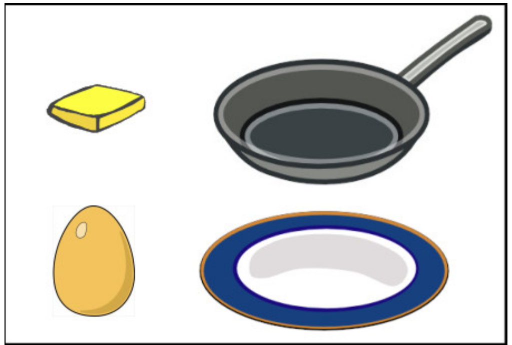


*egg=whole,*
*butter_in=table,*
*egg_in=table*

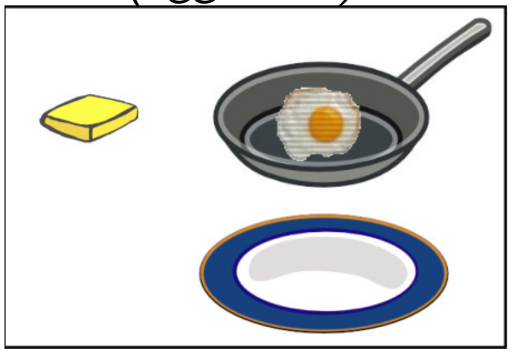


*egg=broken,*
*butter_in=table,*
*egg_in=pan*

**Relational:** PDDL
(What we actually use, not in this course)

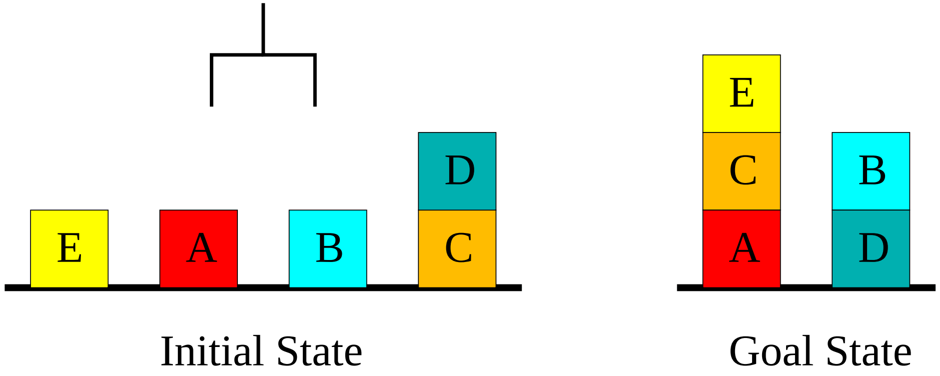


*state(egg,whole),*
*in(butter,table),*
*in(egg,table)*



*state(egg,broken),*
*in(butter,table),*
*in(egg,pan)*

# The Blocksworld in PDDL (STRIPS):



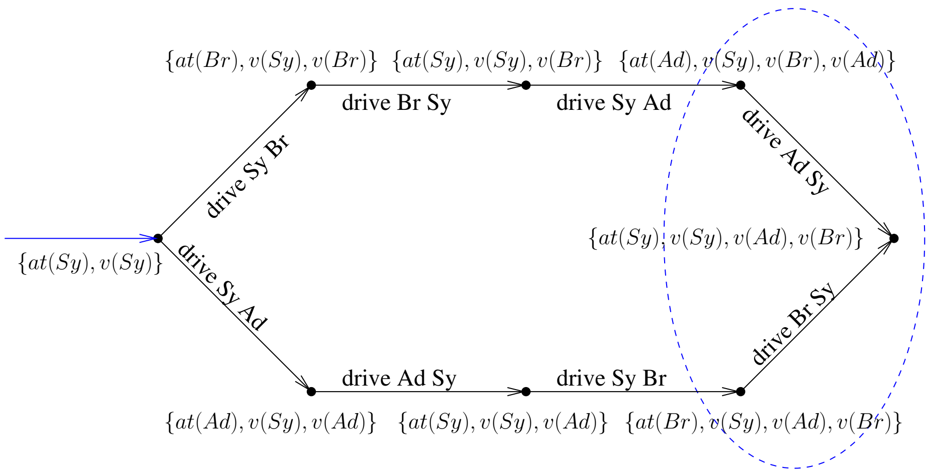

## Domain File:

```
(define (domain blocksworld)
(:predicates (clear ?x) (holding ?x) (on ?x ?y)
             (on-table ?x) (arm-empty))
(:action stack
 :parameters (?x ?y)
 :precondition (and (clear ?y) (holding ?x))
 :effect (and (arm-empty) (on ?x ?y)
              (not (clear ?y)) (not (holding ?x)))
)
...
```

## Problem File:

```
(define (problem bw-abcde)
(:domain blocksworld)
(:objects a b c d e)
(:init (on-table a) (clear a)
       (on-table b) (clear b)
       (on-table e) (clear e)
       (on-table c) (on d c) (clear d)
       (arm-empty))
(:goal (and (on e c) (on c a) (on b d))))
```

# So, How to Solve Planning Problems?



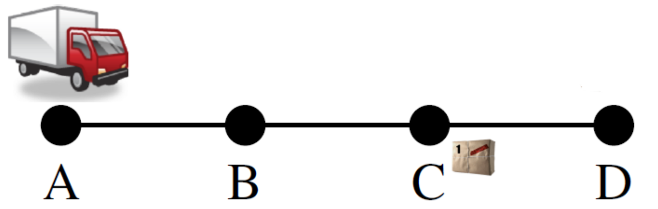

## Question

Given a graph $G$, a node $I$, and a set of nodes $G$, find the shortest path from $I$ to any node in $G$. What algorithm do you suggest to use?

## The Problem

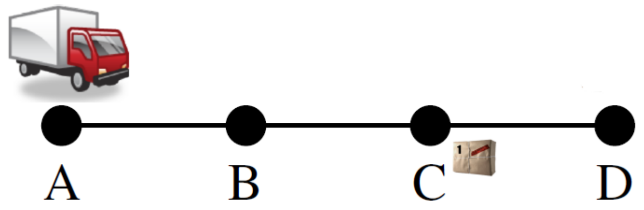
# Example: Planning as Search

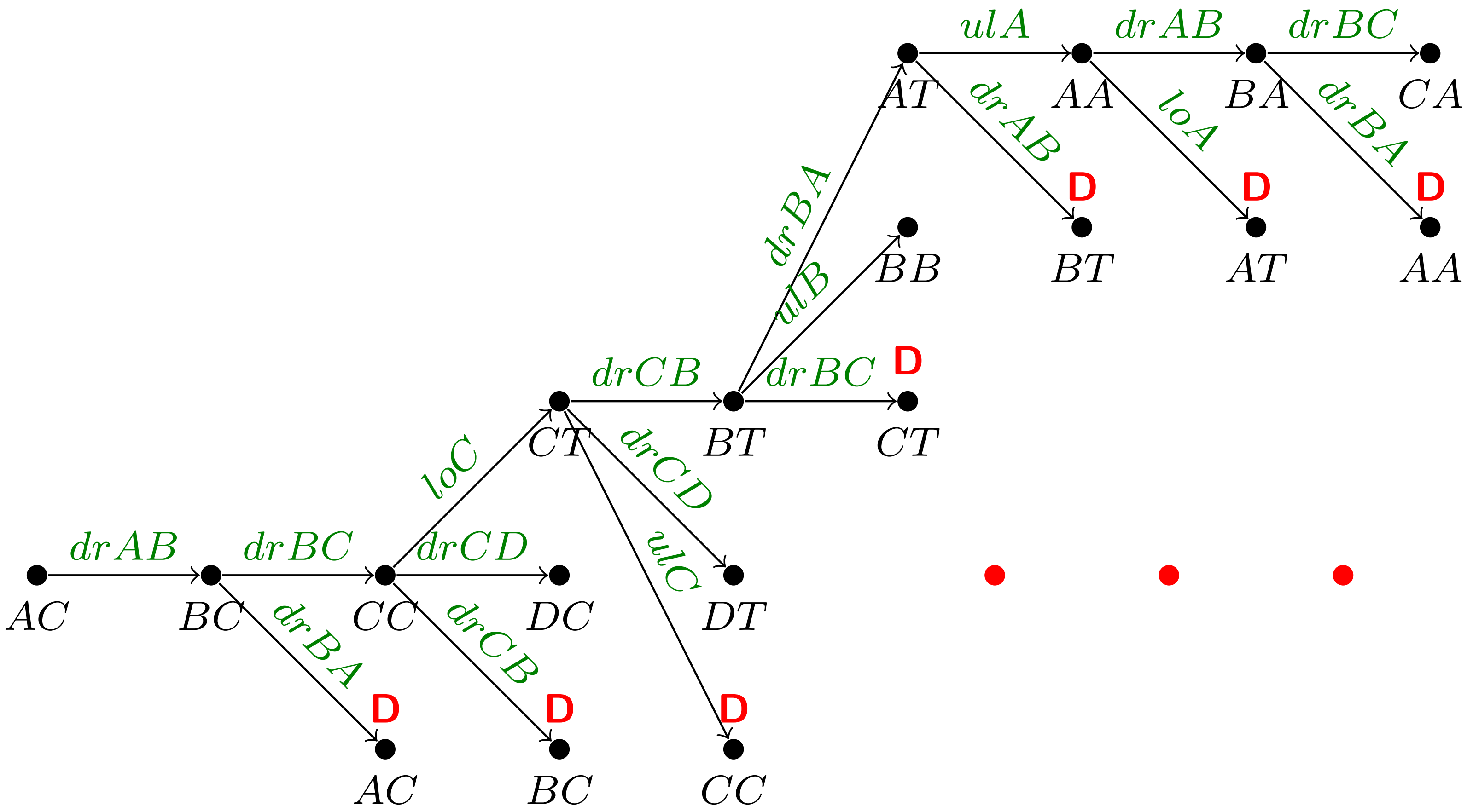
**Example:** "Logistics"



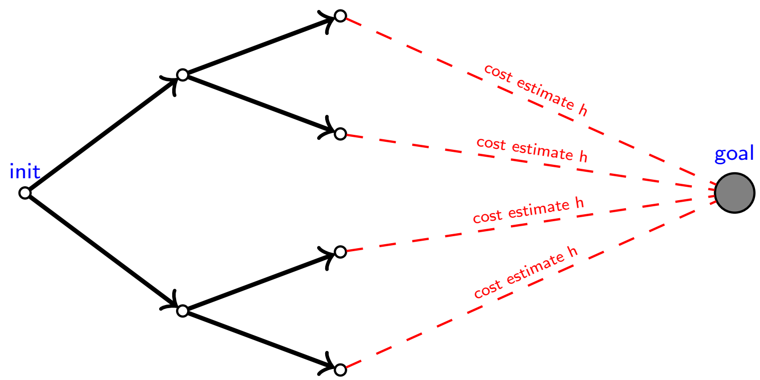

- **Facts $P$:** $\{truck(x) \mid x \in \{A, B, C, D\}\} \cup \{pack(x) \mid x \in \{A, B, C, D, T\}\}$.
- **Initial state $I$:** $\{truck(A),\ pack(C)\}$.
- **Goal $G$:** $\{truck(A),\ pack(D)\}$.
- **Actions $A$:** (Notated as "precondition $\Rightarrow$ adds, $\neg$ deletes")
  - $drive(x, y)$, where $x, y$ have a road: "$truck(x) \Rightarrow truck(y), \neg truck(x)$".
  - $load(x)$: "$truck(x), pack(x) \Rightarrow pack(T), \neg pack(x)$".
  - $unload(x)$: "$truck(x), pack(T) \Rightarrow pack(x), \neg pack(T)$".

# Example: Planning as Search



# Heuristic Search




→ Heuristic function $h$ estimates the cost of an optimal path from a state $s$ to the goal; search prefers to expand states $s$ with small $h(s)$.

**Definition (Heuristic Function).** *Let $\Pi$ be a planning task with state space $\Theta_\Pi = (S, L, c, T, I, S^G)$. A* **heuristic function**, *short* **heuristic**, *for $\Pi$ is a function $h : S \mapsto \mathbb{R}_0^+ \cup \{\infty\}$. Its value $h(s)$ for a state $s$ is referred to as the state's* **heuristic value**, *or $h$* **value**.

**Definition (Remaining Cost, $h^*$).** *Let $\Pi$ be a planning task with state space $\Theta_\Pi = (S, L, c, T, I, S^G)$. For a state $s \in S$, the state's* **remaining cost** *is the cost of an optimal plan for $s$, or $\infty$ if there exists no plan for $s$. The* **perfect heuristic** *for $\Pi$, written $h^*$, assigns every $s \in S$ its remaining cost as the heuristic value.*

→ Heuristic functions $h$ estimate remaining cost $h^*$.

# Properties of Individual Heuristic Functions

**Definition (Safe/Goal-Aware/Admissible/Consistent).** *Let $\Pi$ be a planning task with state space $\Theta_\Pi = (S, L, c, T, I, S^G)$, and let $h$ be a heuristic for $\Pi$. The heuristic is called:*

- **safe** *if, for all $s \in S$, $h(s) = \infty$ implies $h^*(s) = \infty$;*
- **goal-aware** *if $h(s) = 0$ for all goal states $s \in S^G$;*
- **admissible** *if $h(s) \leq h^*(s)$ for all $s \in S$;*
- **consistent** *if $h(s) \leq h(s') + c(a)$ for all transitions $s \xrightarrow{a} s'$.*

$\rightarrow$ **Relationships:**

**Proposition.** *Let $\Pi$ be a planning task, and let $h$ be a heuristic for $\Pi$. Then:*

- *If $h$ is admissible, then $h$ is goal-aware.*
- *If $h$ is admissible, then $h$ is safe.*
- *If $h$ is consistent and goal-aware, then $h$ is admissible.*
- *No other implications of this form hold.*

# Greedy Best-First Search and $\mathrm{A}^*$

**Duplicate elimination omitted**

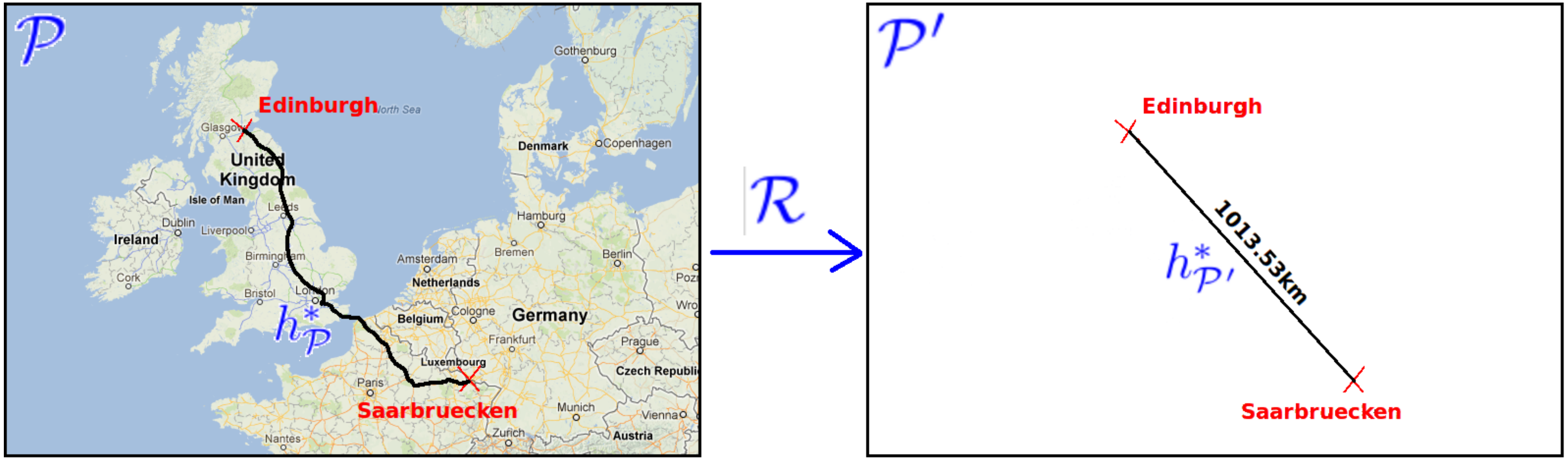
**function** Greedy Best-First Search **[A$^*$]**(*problem*) **returns** a solution, or failure
  *node* ← a node $n$ with $n$.*state*=*problem.InitialState*
  *frontier* ← a priority queue ordered by ascending $h$ [$g+h$], only element $n$
  **loop do**
    **if** *Empty?(frontier)* **then return** failure
    $n$ ← *Pop(frontier)*
    **if** *problem.GoalTest*($n$.*State*) **then return** *Solution*($n$)
    **for each** *action* $a$ **in** *problem.Actions*($n$.*State*) **do**
      $n'$ ← *ChildNode(problem,*$n$,$a$)
      *Insert*($n'$, $h(n')$ [$g(n')+h(n')$], *frontier*)

→ Greedy best-first search explores states by increasing heuristic value $h$. $\mathrm{A}^*$ explores states by increasing plan-cost estimate $g+h$.

**Greedy best-first search: Fast but not optimal $\Longrightarrow$ satisficing planning.**

**$\mathrm{A}^*$: Optimal for admissible $h$ $\Longrightarrow$ optimal planning, with such $h$.**

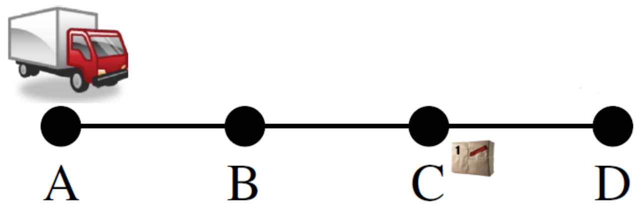
# Relaxation in Route-Finding




- **Problem class $\mathcal{P}$:** Route finding.
- **Perfect heuristic $h^*_{\mathcal{P}}$ for $\mathcal{P}$:** Length of a shortest route.
- **Simpler problem class $\mathcal{P}'$:**
- **Perfect heuristic $h^*_{\mathcal{P}'}$ for $\mathcal{P}'$:**
- **Transformation $\mathcal{R}$:**

# How to Relax in Planning? (A Reminder!)

**Example:** "Logistics"




- **Facts $P$:** $\{truck(x) \mid x \in \{A,B,C,D\}\} \cup \{pack(x) \mid x \in \{A,B,C,D,T\}\}$.
- **Initial state $I$:** $\{truck(A), \; pack(C)\}$.
- **Goal $G$:** $\{truck(A), \; pack(D)\}$.
- **Actions $A$:** (Notated as "precondition $\Rightarrow$ adds, $\neg$ deletes")
  - $drive(x,y)$, where $x,y$ have a road: "$truck(x) \Rightarrow truck(y), \neg truck(x)$".
  - $load(x)$: "$truck(x), pack(x) \Rightarrow pack(T), \neg pack(x)$".
  - $unload(x)$: "$truck(x), pack(T) \Rightarrow pack(x), \neg pack(T)$".

**Example "Only-Adds" Relaxation: Drop the preconditions and deletes.**

"$drive(x,y)$: $\Rightarrow truck(y)$"; "$load(x)$: $\Rightarrow pack(T)$"; "$unload(x)$: $\Rightarrow pack(x)$".

$\rightarrow$ **Heuristic value for $I$ is?**

# How to Relax During Search: Overview

**Attention! Search uses the real (un-relaxed) $\Pi$. The relaxation is applied (e.g., in Only-Adds, the simplified actions are used) only within the call to $h(s)$!!!**

- **A common student mistake is to instead apply the relaxation once to the whole problem, then doing the whole search "within the relaxation".**
- The next slide illustrates the correct search process in detail.

# How to Relax During Search: Only-Adds

## How the Delete Relaxation Changes the World

# The Delete Relaxation

**Definition (Delete Relaxation).** *Let $\Pi = (P, A, I, G)$ be a planning task. The* **delete-relaxation** *of $\Pi$ is the task $\Pi^+ = (P, A^+, I, G)$ where $A^+ = \{a^+ \mid a \in A\}$ with $pre_{a^+} = pre_a$, $add_{a^+} = add_a$, and $del_{a^+} =$*

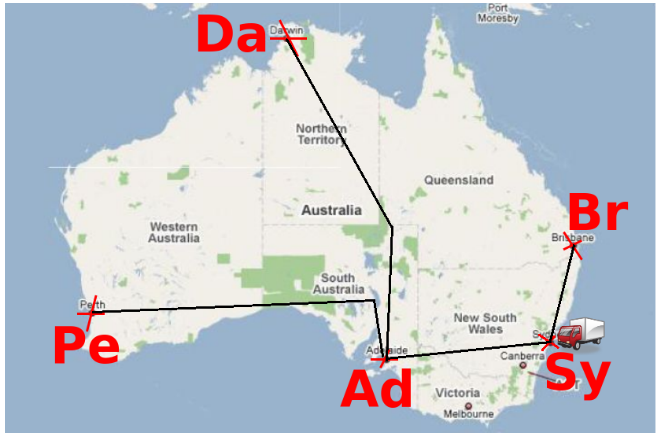
→ In other words, the class of simpler problems $\mathcal{P}'$ is the set of all STRIPS planning tasks with empty delete lists, and the relaxation mapping $\mathcal{R}$ drops the delete lists.

**Definition (Relaxed Plan).** *Let $\Pi = (P, A, I, G)$ be a planning task, and let $s$ be a state. A* **relaxed plan** *for $s$ is a plan for . A relaxed plan for $I$ is called a relaxed plan for $\Pi$.*

→ A relaxed plan for $s$ is an action sequence that solves $s$ when pretending that all delete lists are empty.

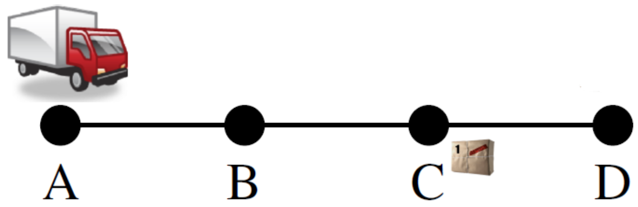
→ Also called **delete-relaxed plan**; "relaxation" is often used to mean "delete-relaxation" by default.

# A Relaxed Plan for "TSP" in Australia




1. **Initial state:** $\{at(Sydney), visited(Sydney)\}$.
2. **Apply** $drive(Sydney, Brisbane)^+$**:** $\{at(Brisbane), visited(Brisbane), at(Sydney), visited(Sydney)\}$.
3. **Apply** $drive(Sydney, Adelaide)^+$**:** $\{at(Adelaide), visited(Adelaide), at(Brisbane), visited(Brisbane), at(Sydney), visited(Sydney)\}$.
4. **Apply** $drive(Adelaide, Perth)^+$**:** $\{at(Perth), visited(Perth), at(Adelaide), visited(Adelaide), at(Brisbane), visited(Brisbane), at(Sydney), visited(Sydney)\}$.
5. **Apply** $drive(Adelaide, Darwin)^+$**:** $\{at(Darwin), visited(Darwin), at(Perth), visited(Perth), at(Adelaide), visited(Adelaide), at(Brisbane), visited(Brisbane), at(Sydney), visited(Sydney)\}$.

# A Relaxed Plan for "Logistics"




- **Facts $P$:** $\{truck(x) \mid x \in \{A, B, C, D\}\} \cup pack(x) \mid x \in \{A, B, C, D, T\}\}$.
- **Initial state $I$:** $\{truck(A),\ pack(C)\}$.
- **Goal $G$:** $\{truck(A),\ pack(D)\}$.
- **Relaxed actions $A^+$:** (Notated as "precondition $\Rightarrow$ adds")
  - $drive(x, y)^+$: "$truck(x) \Rightarrow truck(y)$".
  - $load(x)^+$: "$truck(x), pack(x) \Rightarrow pack(T)$".
  - $unload(x)^+$: "$truck(x), pack(T) \Rightarrow pack(x)$".

**Relaxed plan:**

# PlanEx$^+$

**Definition (Relaxed Plan Existence Problem).** *By* **PlanEx$^+$**, *we denote the problem of deciding, given a planning task* $\Pi = (P, A, I, G)$, *whether or not there exists a* **relaxed plan** *for* $\Pi$.

→ **This is easier than PlanEx for general STRIPS!**

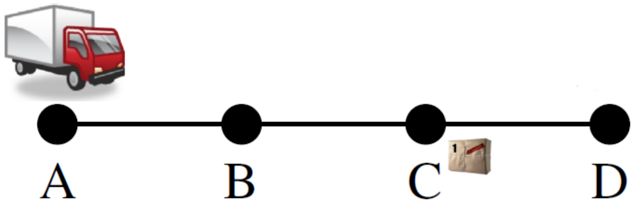
**Proposition (PlanEx$^+$ is Easy).** *PlanEx$^+$ is a member of* **P**.

**Proof.** The following algorithm decides PlanEx$^+$:

$F := I$
**while** $G \not\subseteq F$ **do**
  $F' := F \cup \bigcup_{a \in A: pre_a \subseteq F} add_a$
  (*) **if** $F' = F$ **then return** "unsolvable" **endif**
  $F := F'$
**endwhile**
**return** "solvable"

The algorithm terminates after at most

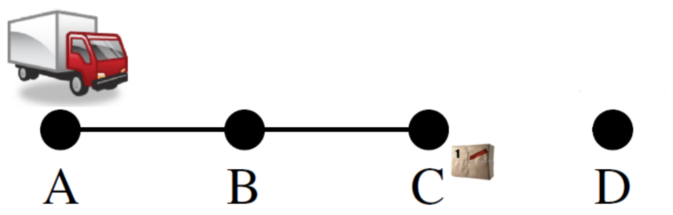
# Deciding PlanEx$^+$ in "Logistics"




**Iterations on $F$:**

- $\{truck(A), pack(C)\}$
- $\cup$
- $\cup$
- $\cup$
- $\cup$

# Deciding PlanEx$^+$ in Unsolvable "Logistics"




**Iterations on $F$:**

- $\{truck(A), pack(C)\}$
- $\cup$
- $\cup$
- $\cup$
- $\cup$
- $\cup$

## Questionnaire

### Question!

**How does ignoring delete lists simplify Sokoban?**

**(A):** You will never "lock yourself in".

**(B):** Free positions remain free.

**(C):** You can walk through walls.

**(D):** A single action can push 2 stones at once.

# The $h^+$ Heuristic

$\rightarrow$ **PlanEx$^+$ is not actually what we're looking for. PlanEx$^+$ $=$ relaxed plan existence; we want relaxed plan length $h^* \circ \mathcal{R}$.**

**Definition (Optimal Relaxed Plan).** *Let $\Pi = (P, A, I, G)$ be a planning task, and let $s$ be a state. An* **optimal relaxed plan** *for $s$ is an optimal plan for $(P, A, s, G)^+$.*

**Here's what we're looking for:**

**Definition ($h^+$).** *Let $\Pi = (P, A, I, G)$ be a planning task with states $S$. The* **ideal delete-relaxation heuristic** $h^+$ *for $\Pi$ is the function $h^+ : S \mapsto \mathbb{N}_0 \cup \{\infty\}$ where $h^+(s)$ is the length of an optimal relaxed plan for $s$ if a relaxed plan for $s$ exists, and $h^+(s) = \infty$ otherwise.*

$\rightarrow$ In other words, $h^+ = h^* \circ \mathcal{R}$, cf. previous slide.

# $h^+$ is Admissible

**Lemma.** *Let $\Pi = (P, A, I, G)$ be a planning task, and let $s$ be a state. If $\langle a_1, \dots, a_n \rangle$ is a plan for $(P, A, s, G)$, then $\langle a_1^+, \dots, a_n^+ \rangle$ is a plan for $(P, A, s, G)^+$.*

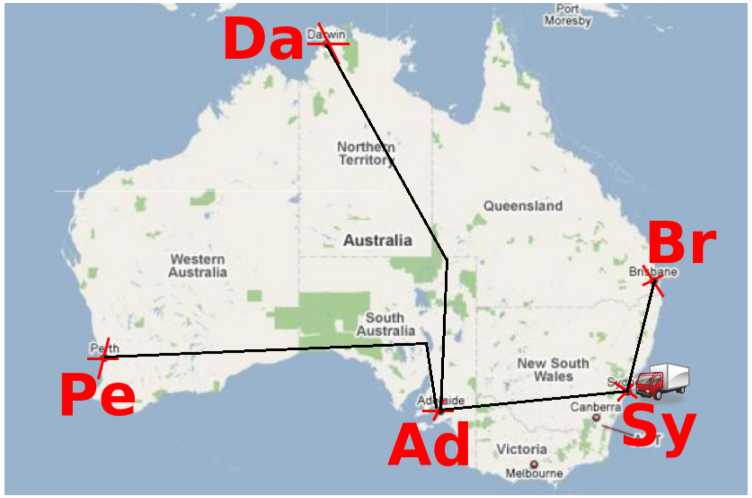
**Proof Sketch. (for reference)** Show by induction over $0 \leq i \leq n$ that $appl(s, \langle a_1, \dots, a_i \rangle) \subseteq appl(s, \langle a_1^+, \dots, a_i^+ \rangle)$.

$\rightarrow$ **"If we ignore deletes, the states along the plan can only get bigger."**

**Theorem.** *$h^+$ is Admissible.*

**Proof. (for reference)** Let $\Pi = (P, A, I, G)$ be a planning task with states $S$, and let $s \in S$. $h^+(s)$ is defined as optimal plan length in $(P, A, s, G)^+$. With the above lemma, any plan for $(P, A, s, G)$ also constitutes a plan for $(P, A, s, G)^+$. Thus optimal plan length in $(P, A, s, G)^+$ cannot be longer than that in $(P, A, s, G)$, and the claim follows.

# $h^+$ in "TSP" in Australia




**Planning vs. Relaxed Planning:**

- **Optimal plan:** $\langle drive(Sydney, Brisbane), drive(Brisbane, Sydney), drive(Sydney, Adelaide), drive(Adelaide, Perth), drive(Perth, Adelaide), drive(Adelaide, Darwin), drive(Darwin, Adelaide), drive(Adelaide, Sydney)\rangle$.
- **Optimal relaxed plan:** $\langle drive(Sydney, Brisbane), drive(Sydney, Adelaide), drive(Adelaide, Perth), drive(Adelaide, Darwin)\rangle$.
- $h^*(I) = 8$; $h^+(I) = 4$.

# How to Relax During Search: Ignoring Deletes

# Approximating $h^+$: $h^{\text{FF}}$

**Theorem.** *PlanLen$^+$ is* **NP***-complete.*

→ We can't compute our heuristic $h^+$ efficiently. So we approximate it instead.

**Definition ($h^{\text{FF}}$).** *Let $\Pi = (P, A, I, G)$ be a planning task with states $S$. A* **relaxed plan heuristic $h^{\text{FF}}$** *for $\Pi$ is a function $h^{\text{FF}} : S \mapsto \mathbb{N}_0 \cup \{\infty\}$ returning* **the length of some, not necessarily optimal, relaxed plan for $s$** *if a relaxed plan for $s$ exists, and returning $h^{\text{FF}}(s) = \infty$ otherwise.*

**Notes:**

- $h^{\text{FF}} \geq h^+$, i.e., $h^{\text{FF}}$ never under-estimates $h^+$.
- **We may have $h^{\text{FF}} > h^*$, i.e., $h^{\text{FF}}$ is not admissible!** Thus $h^{\text{FF}}$ can be used for satisficing planning only, not for optimal planning.

**Observe:** **$h^{\text{FF}}$ as per this definition is not unique.** How do we find *"some, not necessarily optimal, relaxed plan for $(P, A, s, G)$"*?

→ In what follows, we consider the following algorithm computing relaxed plans, and therewith (one variant of) $h^{\text{FF}}$:

1. Chain **forward** to build a **relaxed planning graph (RPG)**.
2. Chain **backward** to extract a relaxed plan from the RPG.

# Computing $h^{\mathsf{FF}}$: Relaxed Planning Graphs (RPG)

$F_0 := s$, $t := 0$
**while** $G \not\subseteq F_t$ **do**
    $A_t := \{a \in A \mid pre_a \subseteq F_t\}$
    $F_{t+1} := F_t \cup \bigcup_{a \in A_t} add_a$
    **if** $F_{t+1} = F_t$ **then** stop **endif**
    $t := t + 1$
**endwhile**

$\rightarrow$ **Does this look familiar to you?**

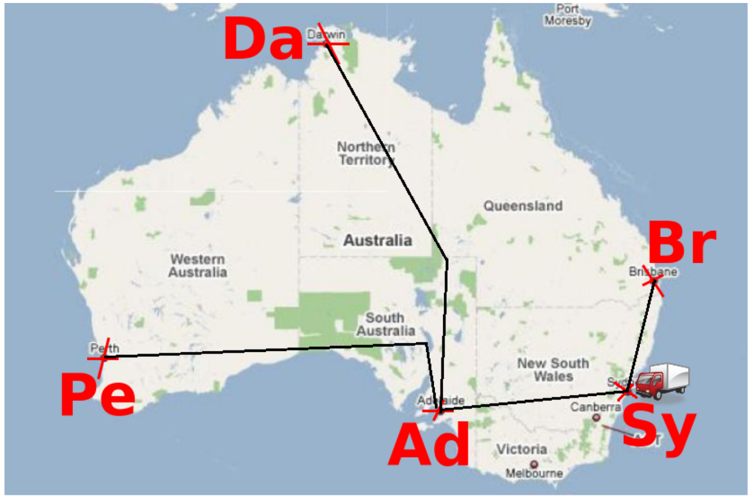
# Computing $h^{\text{FF}}$: Extracting a Relaxed Plan

**Information from the RPG:** (min over an empty set is $\infty$)

- For $p \in P$: $level(\mathbf{p}) := \min\{\mathbf{t} \mid \mathbf{p} \in \mathbf{F_t}\}$.
- For $a \in A$: $level(\mathbf{a}) := \min\{\mathbf{t} \mid \mathbf{a} \in \mathbf{A_t}\}$.

$M := \max\{level(p) \mid p \in G\}$
**If** $M = \infty$ **then** $h^{\text{FF}}(s) := \infty$; stop **endif**
**for** $t := 0, \dots, M$ **do**
    $G_t := \{g \in G \mid level(g) = t\}$
**endfor**
**for** $t := M, \dots, 1$ **do**
    **for** all $g \in G_t$ **do**
        select $a$, $level(a) = t - 1$, $g \in add_a$
        **for** all $p \in pre_a$ **do** $G_{level(p)} := G_{level(p)} \cup \{p\}$ **endfor**
    **endfor**
**endfor**
$h^{\text{FF}}(s) :=$ number of selected actions

# Computing $h^{\text{FF}}$ in "TSP" in Australia




**RPG:**

- $F_0 =$
- $A_0 =$
- $F_1 = F_0 \cup$
- $A_1 = A_0 \cup \quad drive(Adelaide, Sydney),\ drive(Brisbane, Sydney)\}.$
- $F_2 = F_1 \cup$

## Summary

- **Planning** is a form of general problem solving: develop solvers that perform well across a large class of problems.
- STRIPS is the simplest possible, while reasonably expressive, language for our purposes. It uses Boolean variables (facts), and defines actions in terms of precondition, add list, and delete list.
- We will consider Greedy Best-First Search and $\mathrm{A}^*$as heuristic search algorithms.
- Heuristic search on classical search problems relies on a function $h$ mapping states $s$ to an estimate $h(s)$ of their goal distance. Such functions $h$ are derived by solving **relaxed problems**.
- In planning, the relaxed problems are generated and solved automatically.
- The **delete relaxation** consists in dropping the deletes from STRIPS planning tasks. A **relaxed plan** is a plan for such a relaxed task. $h^+(s)$ is the length of an optimal relaxed plan for state $s$.

## On the "Accuracy" of $h^+$

**Reminder:** Heuristics based on ignoring deletes are the key ingredient to almost all winners of the International Planning Competition in the last two decades.

$\rightarrow$ **Why?**

$\rightarrow$ A heuristic function is useful if its estimates are "accurate".

**How to measure this?**

- **Known method 1:** Error relative to $h^*$, i.e., bounds on $|h^*(s) - h(s)|$.
- **Known method 2:** Properties of the **search space surface**: Local minima etc.

$\rightarrow$ **For $h^+$, method 2 is the road to success:**

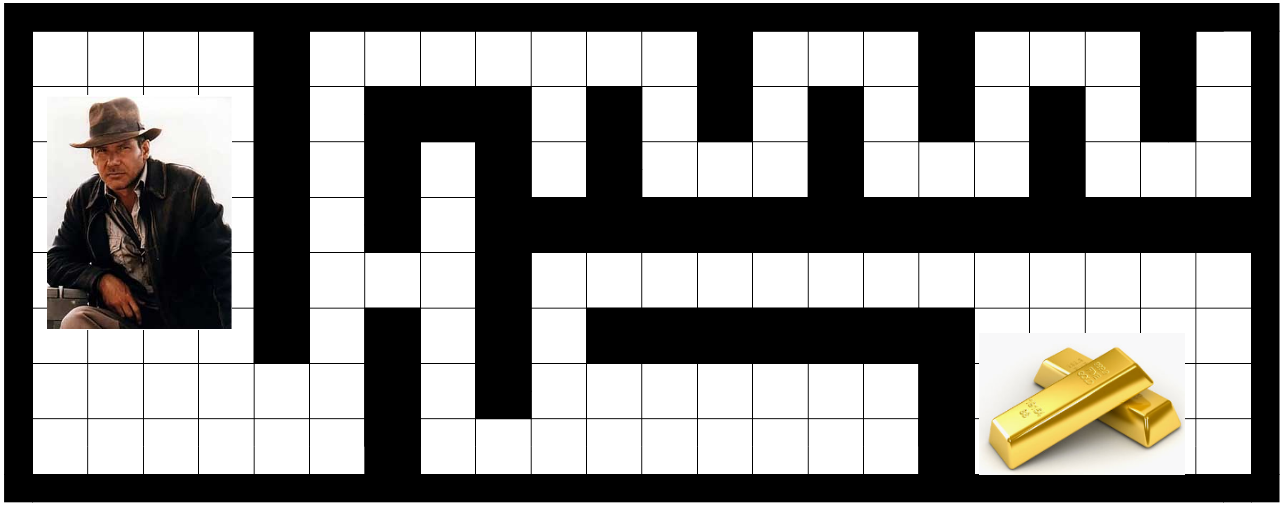
$\rightarrow$ In many benchmarks, under $h^+$, local minima *provably* do not exist! [?]

# A Brief Glimpse of $h^+$ Search Space Surfaces

# $h^+$ in (the Real) TSP

# $h^+$ in Graphs

# Questionnaire



**Question!**

**In this domain, $h^+$ is equal to?**

**(A):** Manhattan Distance. **(B):** Horizontal distance.

**(C):** Vertical distance. **(D):** $h^*$.