Linköping University

Automated Planning

Introduction to Planning

Jonas Kvarnström
Automated Planning Group
Department of Computer and Information Science
Linköping University

jonas.kvarnstrom@liu.se – 2018

<u>**One**</u> way of defining planning:

*Using **<u>knowledge</u>** about the world,*
including possible actions and their results,
*to **<u>decide</u>** what to do and when*
*in order to achieve an **<u>objective</u>**,*
***<u>before</u>** you actually start doing it*

Applications of Planning

Towers of Hanoi

Some applications are simple, well-structured, almost *toy problems*

Simple structure

Single agent acting

Possible actions

- **Move** topmost disk from *x* to *y*, **without** placing larger disks on smaller disks

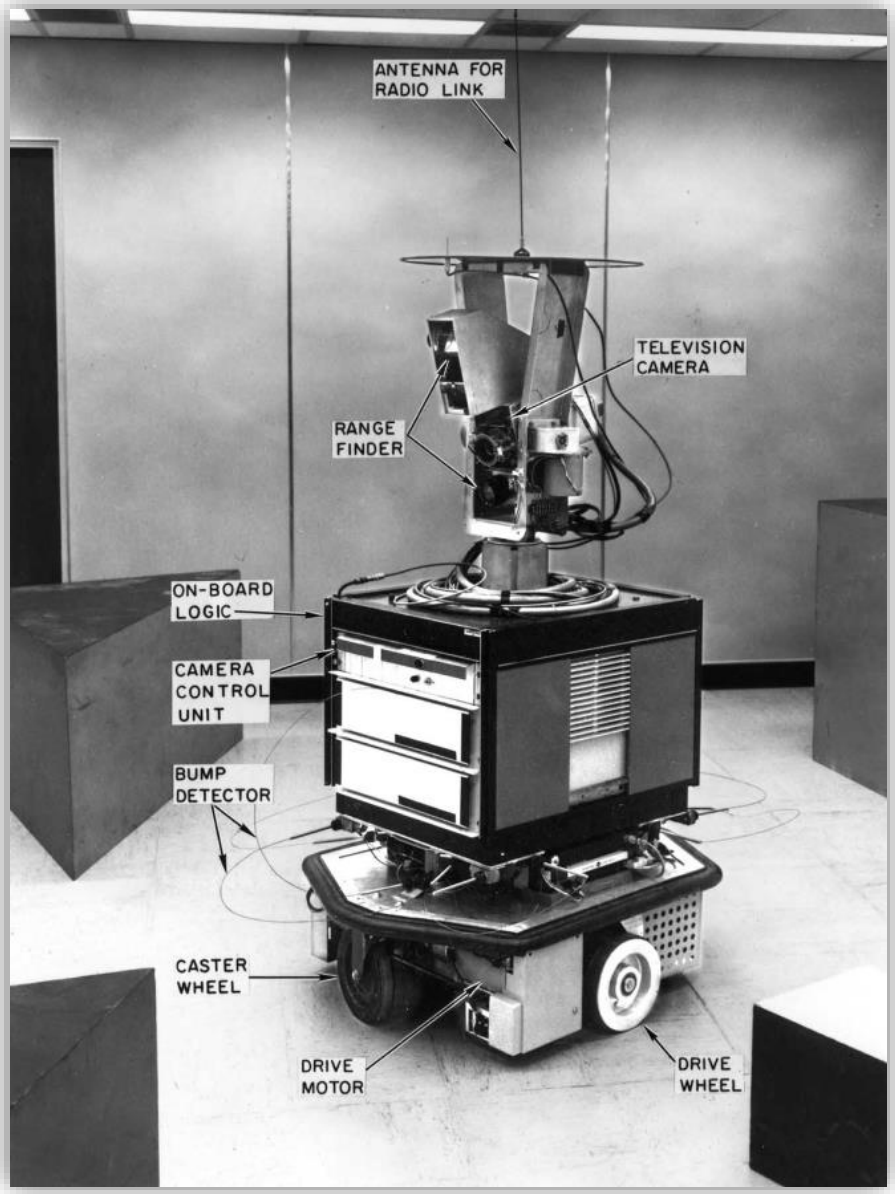
Objective

- All disks on the *third* peg, in order of increasing size

Shakey

- Classical robot example: **Shakey** (1969)
  - Available **actions**:
    - *Moving* to another location
    - *Turning* light switches on and off
    - *Opening* and *closing* doors
    - *Pushing* movable objects around
    - …
  - **Goals**:
    - *Be in room 4 with objects A,B,C*

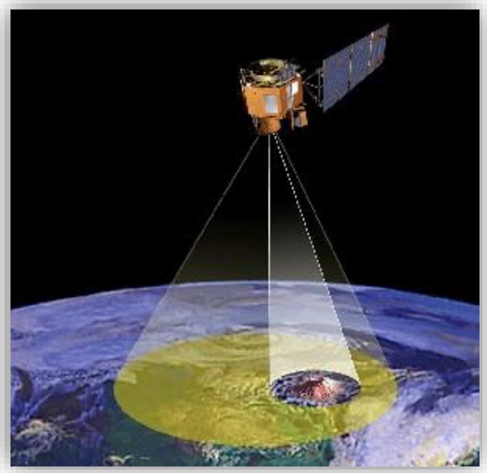
    - http://www.youtube.com/watch?v=qXdn6ynwpiI



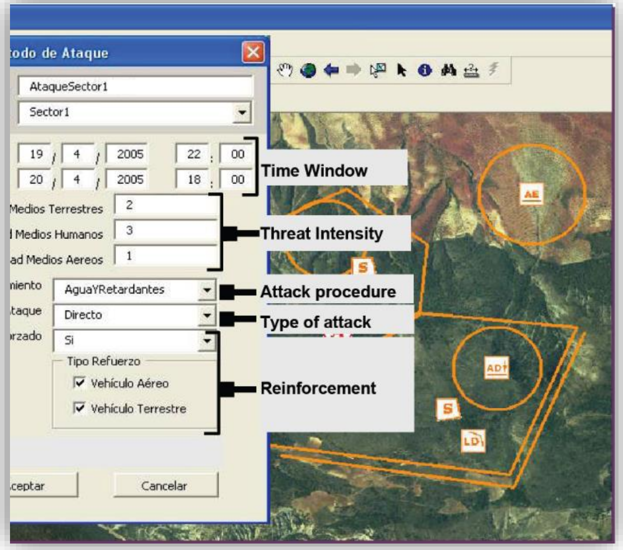
Miconic 10 Elevators

- **Schindler Miconic 10 system**
  - Tall buildings, multiple elevators
  - Enter destination *before* you board
  - System creates a *plan*:
    - Which elevator goes to which floor
    - In which order
  - Saves time!
    - 3 elevators could serve as much traffic as 5 elevators with earlier algorithms

Earth and Space



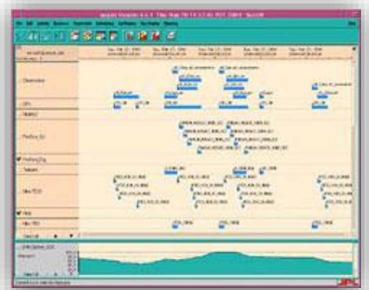
On-board planning
to view interesting natural events:
http://ase.jpl.nasa.gov/



odo de Ataque

AtaqueSector1
Sector1
19 / 4 / 2005 22 : 00
20 / 4 / 2005 18 : 00
Medios Terrestres 2
Medios Humanos 3
ad Medios Aereos 1
niento AguaYRetardantes
taque Directo
rzado Si
Tipo Refuerzo
☑ Vehículo Aéreo
☑ Vehículo Terrestre
ceptar Cancelar

SIADEX –
plan for firefighting
Limited resources
Plan execution is dangerous!



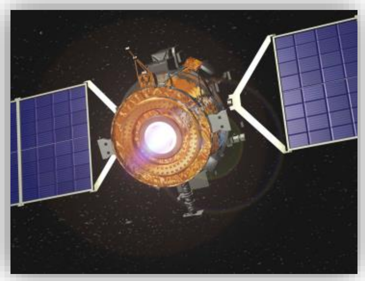
NASA Mapgen / Mars Rovers

Primary platform for creating daily activity plans for Spirit, Opportunity

Mixed-initiative tool:
Human in the loop

Why should <u>Computers</u> Plan?

- And why should **<u>computers</u>** create plans?
  - Manual planning can be **<u>boring</u>** and **<u>inefficient</u>**
  - Automated planning may **<u>create higher quality plans</u>**
    - Software can systematically optimize
  - Automated planning can be applied **<u>where the agent is</u>**
    - Satellites cannot always communicate with ground operators
    - Spacecraft or robots on other planets may be hours away by radio



Distinction:
Planning vs. Reacting

Context: Unmanned Aerial Vehicles

- A modern **context** for planning:
  - Autonomous
    Unmanned Aerial Vehicles (UAVs)

*Using **knowledge** about the world,*
including possible actions and their results,
*to **decide** what to do and when*
*in order to achieve an **objective**,*
***before** you actually start doing it*

Actions for UAVs

- General knowledge about the world
  - **Locations** of UAVs and objects
  - **Fuel** levels, …
- Available actions:
  - **Take off**
    - Before: The UAV must be on the ground
    - Result: The UAV is flying
  - **Fly to a point**
    - Before: Must have sufficient fuel
    - Result: Will end up at the indicated point
  - **Land**
  - **Fly a trajectory curve**
  - **Point camera, take picture**
  - **Start/stop video recording**
  - **Transmit information to operator**
  - **…**

Informal!
Incomplete!

More later…

Using ***knowledge*** *about the world,*

including possible actions and their results,

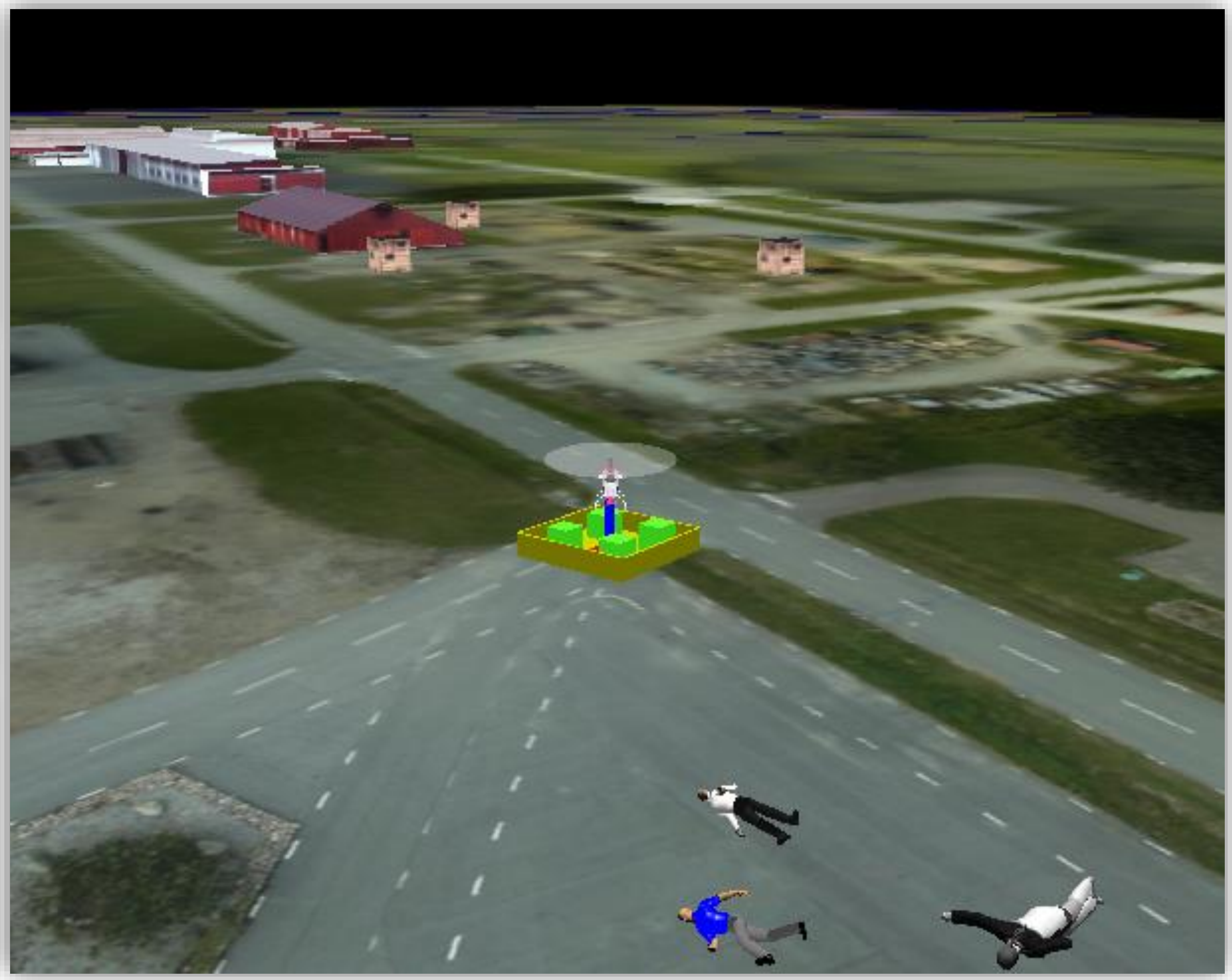
to ***decide*** *what to do and when*

in order to achieve an ***objective****,*

before *you actually start doing it*

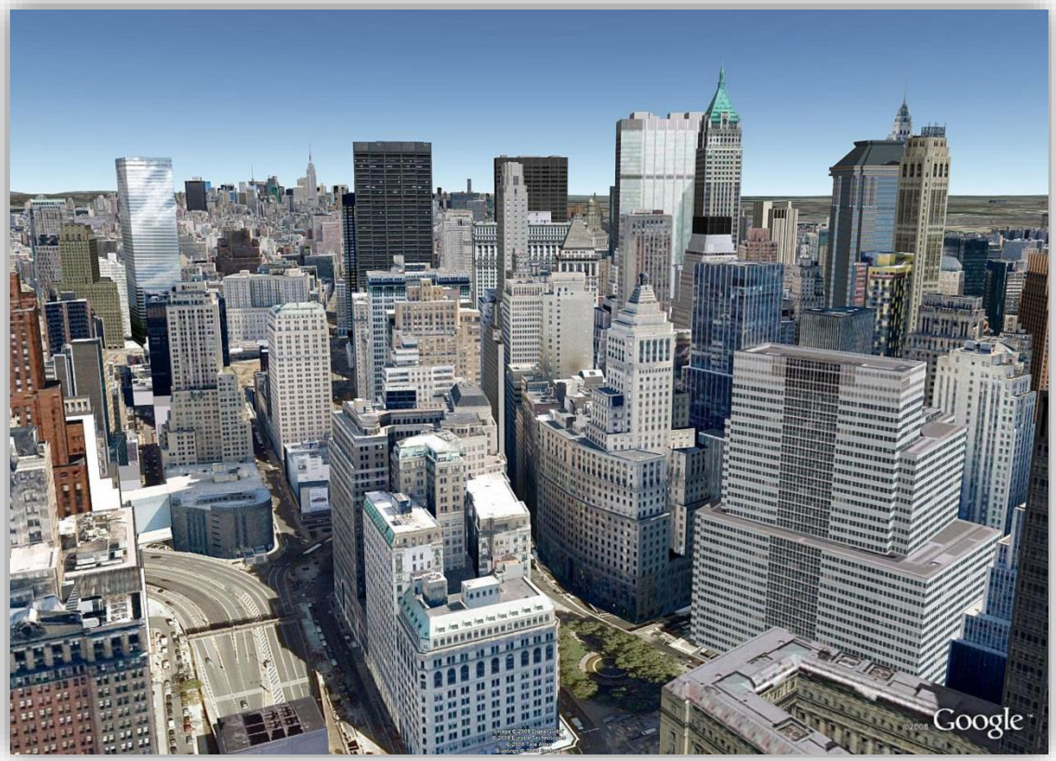
UAV Objective 1: Emergency Services Logistics

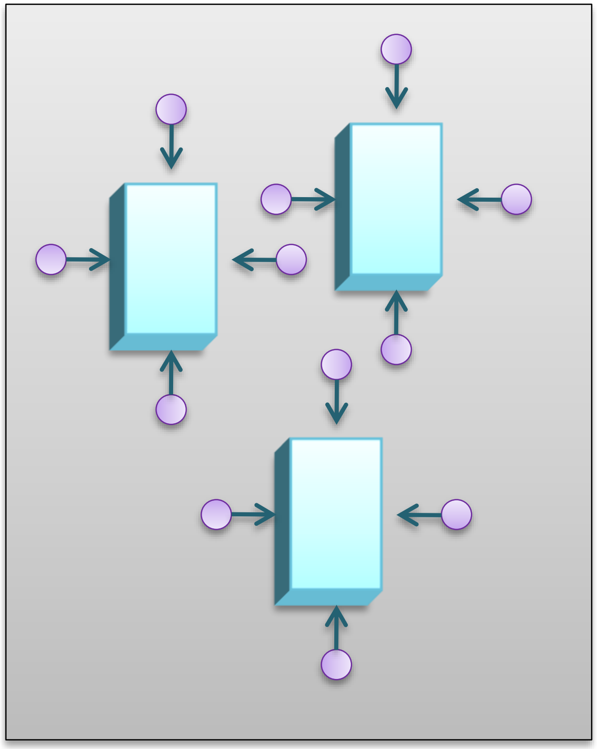
- Assist in emergency situations
  - Deliver packages of food, medicine, water



UAV Objective 2: Photogrammetry

- A specific **photogrammetry problem** with a single UAV:
  - Photograph buildings – generate realistic 3D models
  - Problem: Find best way of taking pictures
    - From specificed locations
    - In the specified directions





*Using **knowledge** about the world,*
including possible actions and their results,
*to **decide** what to do and when*
*in order to achieve an **objective**,*
***before** you actually start doing it*

Method 0: Reactive + Stupid

- Method 0: Let's be **reactive** and **stupid**
  - Reactive: *No planning, don't explicitly consider the future*
  - Very fast *decision* + *execution algorithm:*

```
while (exists unvisited position) {
    pos ← some random unvisited position
    flyto(pos)
    aim()
    take-picture()
}
```

  - Somewhat suboptimal for *flight*…

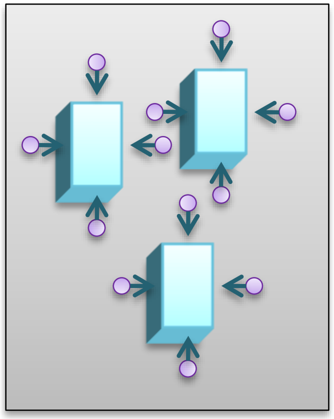
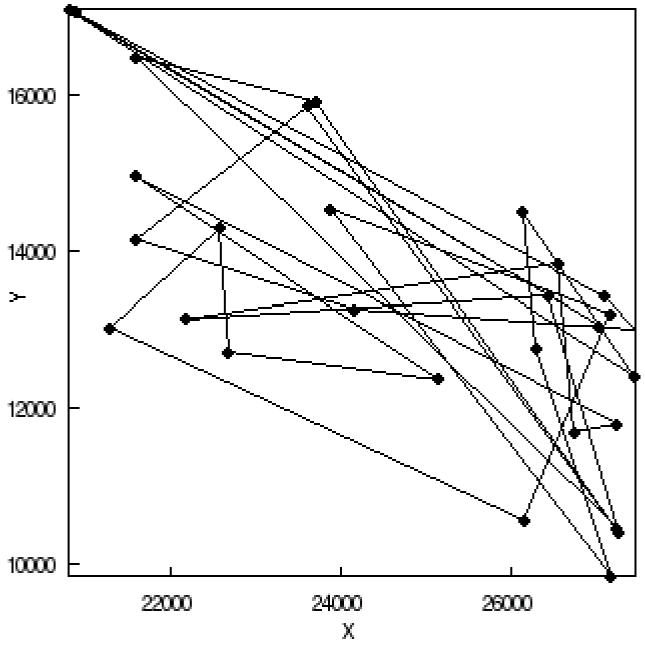


Figure 3.1. Western Sahara: example of random initial tour



http://nfrac.org/felix/publications/tsp/

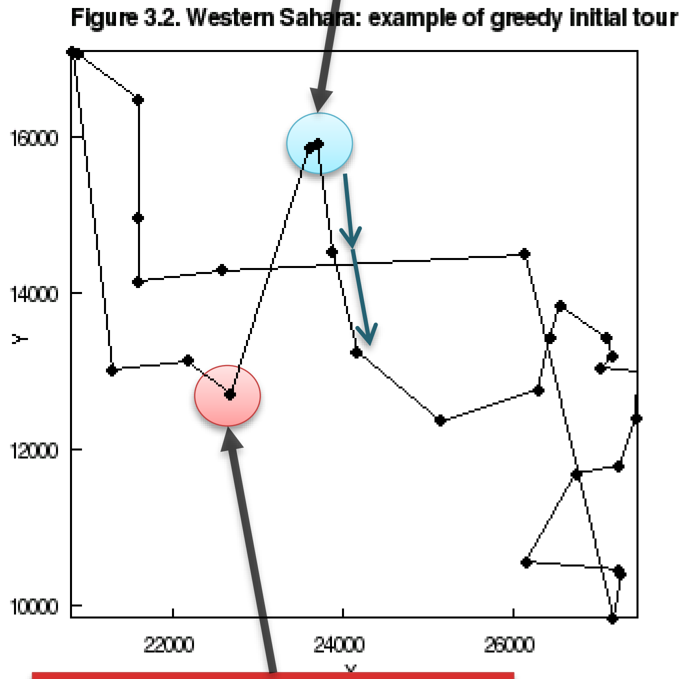
Method 1: Reactive + Greedy

- Method 1: Let's be **reactive** and **greedy**
  - **Greedy heuristic** chooses next location
    - "Least expensive extension to the plan"
  -
```
while (exists unvisited position) {
    pos ← the nearest unvisited position
    flyto(pos)
    aim(); take-picture()
}
```
  - *Seems* good for this task; not optimal
    - Least expensive *right now*, more expensive in the long run
  - For many other tasks: Still *really bad*

Often, *not thinking ahead* means you can't even solve the problem!

(Fly too far ➔ run out of fuel; crack an egg ➔ can't uncrack it; …)

Start here; generate actions incrementally

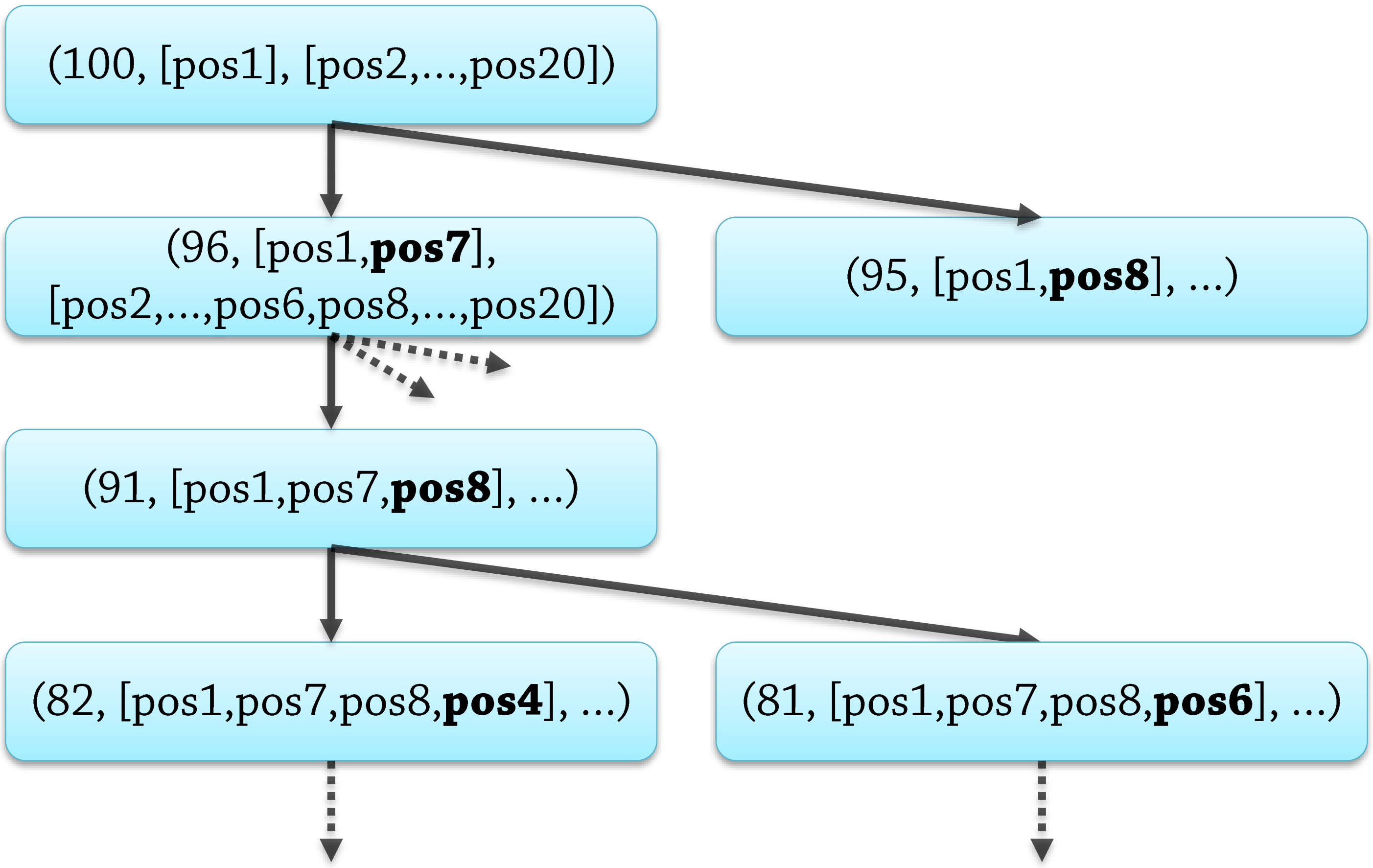


Run out of fuel here?

*Using **knowledge** about the world,*
including possible actions and their results,
*to **decide** what to do and when*
*in order to achieve an **objective**,*
***before** you actually start doing it*

Method 2: Think ahead

- Method 2: Let's **think ahead**
  - *First* create a complete plan, considering multiple choices
  - Keeping track of [**fuel** left, **visited** positions, **remaining** positions]



(100, [pos1], [pos2,...,pos20])
(96, [pos1,**pos7**], [pos2,...,pos6,pos8,...,pos20])
(95, [pos1,**pos8**], ...)
(91, [pos1,pos7,**pos8**], ...)
(82, [pos1,pos7,pos8,**pos4**], ...)
(81, [pos1,pos7,pos8,**pos6**], ...)

Method 2: Think ahead – planning

- Method 2, *first step*: **Search** (for example, depth first)
  - **call solve**(100, [pos1], {pos2,…,pos20})
  - **solve**(*fuel-before*, *plan*, *remaining*) {

```
    if (remaining == ∅) return plan;
    currpos = last(plan)
    foreach position nextpos in remaining
                in order of increasing
                distance(currpos, nextpos)
    {
        after = fuel-before – fuel-usage(currpos, nextpos);
        if (after > 0) {
            plan2 = solve(after, plan+[nextpos],
                            remaining–[nextpos])
            if (plan2 ≠ null) return plan2;
        }
    }
    return null;
}
```

Have we already achieved the goal?

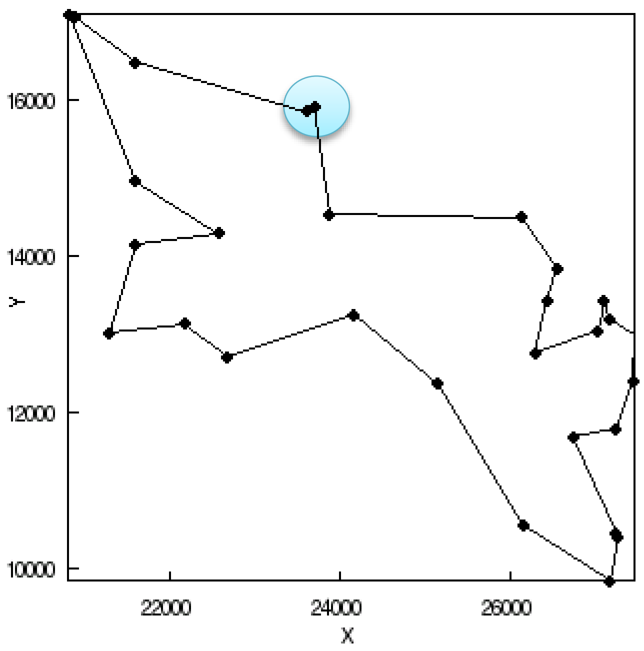
First choice: As before (*greedy heuristic*)
If not feasible: Try the *next nearest* pos
Usually won't have to try *each* position

Check fuel "in simulation", not in reality

Backtrack if there is no feasible continuation

This is (one form of) planning!

Figure 3.8. Western Sahara: solution tour



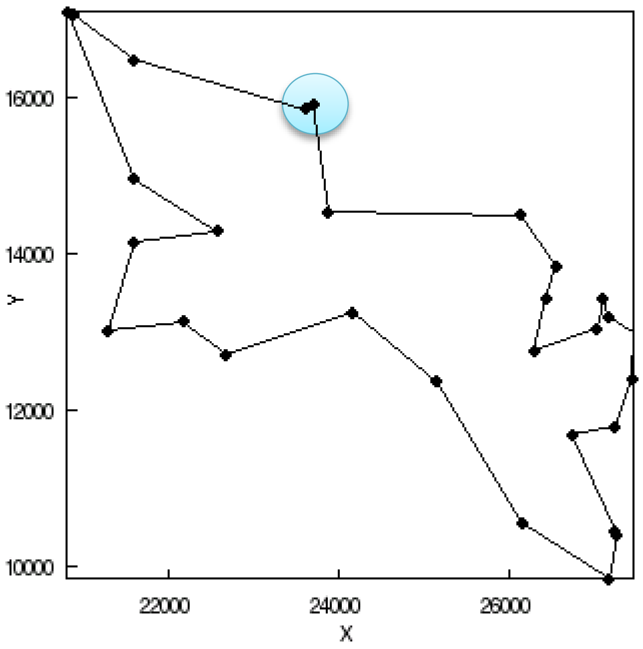
Method 2: Think ahead – execution

- Method 2: Let's think ahead – *second step*, execute the plan

```
▪ seq = solve(100, [pos1], [pos2,…,pos20])
▪ if seq == null:
      signal error
▪ foreach (position pos in seq) {
      flyto(pos)
      aim()
      take-picture()
  }
```

Execution <u>after</u> verifying the plan!

Figure 3.8. Western Sahara: solution tour



Distinction:
Domain-specific vs. domain-independent planning

Domain-Specific Planning 1

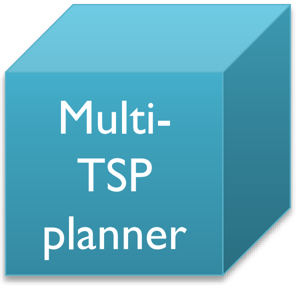
- Our solver is **domain-specific** – only photogrammetry
  - **Strong assumptions**:
    - Interesting aspects of the world:
      fuel left, **visited** positions, **remaining** positions
    - Objective:
      Take a **single** picture at **every** position (no more, no less)
    - Available actions:
      flyto, aim, take-picture
      (executed *in that order* at *each position*)
    - Executability conditions:
      Having **fuel**, being in the right **place**
      (no more, no less)
  - **Positive**: Allows efficient code
    - *Adapted* to the exact problem, "hardcoded"
    - Can even use Traveling Salesman algorithms…

```
solve(fuel-before, plan, remaining) {
  if (remaining == ∅) return order;
    currpos = last(plan)
    foreach position nextpos in remaining
                in order of increasing
                distance(currpos, nextpos)
    {
      after = fuel-before – fuel-usage(currpos, nextpos);
      if (after > 0) {
        plan2 = solve(after, plan+[nextpos],
                           remaining–[nextpos])
        if (plan2 ≠ null) return plan2;
      }
    }
    return null;
}
```

Domain-Specific Planning 2

- But some domains are **less straight-forward!**
  - Writing an efficient solver from scratch is *difficult*
- Specialization means **less flexibility!** What if…
  - you want to **deliver** a couple of crates at the same time?
    - Need to modify the code of the planner

  - you have **two UAVs** and a **UGV** (ground vehicle)?
    - Different algorithm: *Multiple TSP*



  - you want to survey an **area** (send video feed of the ground)?
  - you have dynamic no-fly-zones ("don't fly there at 15:00-16:00")?

Generalization

Many hardcoded assumptions

Interesting aspects of the world:
fuel left, **visited** positions, **remaining** positions

Objective:
Take a **<u>single</u>** picture at **<u>every</u>** position (no more, no less)

Available actions:
flyto, aim, take-picture
(executed *in that order* at *each position*)

Executability conditions:
Having **fuel**, being in the right **place**
(no more, no less)

Little input

Initial fuel level
List of positions

Few assumptions

???

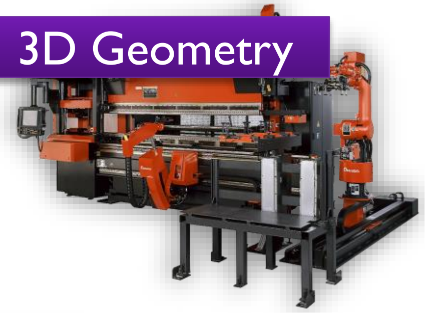
As much as possible specified in the input

?? (should *define* <u>fuel</u>, <u>taking pictures</u>, …)

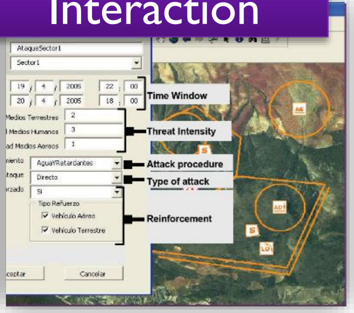
Can we find a **single** set of **common modeling concepts** sufficient for **all** of these **very different** domains?

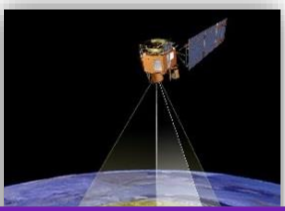
Path planning

3D Geometry



Interaction

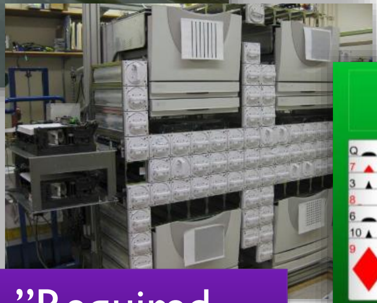




Opportunistic goals

Multiple agents

Simple…



"Required concurrency"

Timing

Can we – and should we?

Different Requirements

Increasing model expressivity:

- Can be **necessary**
  - Can't model fuel usage constraints?
    ➔ create non-executable plans!
- Can **improve quality**
  - Want plans that execute quickly
    ➔ requires a model of time!
- Can **simplify our job**
  - More expressivity
    ➔ easier to express the problem

Decreasing model expressivity:

- Can improve **performance**
  - (By many orders of magnitude)
  - We can exploit problem structure
    - Allows different heuristics
    - Allows different plan structures
    - …
- Simplifies development
- …

Conflicting desires – we need trade-offs!

Decide what "kind" of domains your planner should be able to accept
Write a planner for this **expressivity**

Degrees of Expressivity

"Truly domain-independent"

Handles *everything* – does not exist*

*Except for standard Turing-complete programming languages, which don't count…

Partial order of expressivity, "coverage"…

…

…

Temporal planning

MDP planning

Some classes *subsume* others, some are simply *different*

Classical planning

…

…

Domain-specific (photogrammetry)

Can specialize the planner for very high performance
Must write an entire planner

What is Classical Planning ?

Classical Planning

- Many early planners made **<u>similar tradeoffs</u>**
  - **<u>At the time</u>**, simply called "*planning*" or "*problem solving*"
    - **<u>Later</u>** grouped together, called "**<u>classical planning</u>**"
  - Restricted, but a **<u>good place to start</u>**
    - Forms the basis of most non-classical planners as well

Classical Planning: Disagreements

- So **exactly** what is classical planning?
  - Some disagreements…
    - Inevitable: Just a group of *similar* techniques, formalisms
  - Where to draw the line?

We'll use the book's definitions
You can go outside those boundaries and still be "*kind of classical*"!

Modeling Assumptions
and
State Transition Systems
for classical planning

Intro

Now: Formal classical model

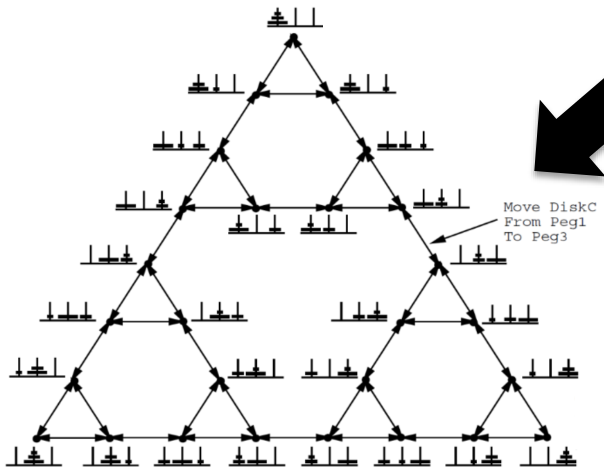
(Σ, s_0, S_g)
where
$\Sigma = (S, A, \gamma)$

**Next time:
Language for describing models**

```
;; The Towers of Hanoi problem
;; (formalisation by Hector Geffner).

(define (domain hanoi)
  (:requirements :strips)
  (:predicates (clear ?x) (on ?x ?y) (smaller ?x ?y))

  (:action move
    :parameters (?disc ?from ?to)
    :precondition (and (smaller ?to ?disc)
                       (on ?disc ?from)
                       (clear ?disc) (clear ?to))
    :effect  (and (clear ?from) (on ?disc ?to)
                  (not (on ?disc ?from))
                  (not (clear ?to))))
  )
```



Classical Planning 1

A0: *Finite* number of states
Can't model continuous positions of disks in ToH
OK – we're only interested in some discrete alternatives:
On peg 1, on peg 2, above disk 3, …

The world is always in a given <u>state</u>, which we want to <u>affect</u>

<u>Photogrammetry</u> state:
fuel left,
visited positions,
remaining positions

- We need **<u>states of the world</u>**
  - $S = \{s_1, s_2, \dots, s_n\}$ is a set of states

Classical Planning 2

A3: The world can only be affected by executing an action

No random changes in the world

No other agents acting in the world

At least not in the part of the world we model!

The world is always in a given state, which we want to affect

- We need **actions**
  - $A = \{a_1, a_2, \dots, a_m\}$ is a set of actions;
    this set is also finite

Classical Planning 3

- We must know **<u>when an action is executable</u>** and **<u>what it achieves</u>**
  - $\gamma : S \times A \rightarrow 2^S$ $\gamma(s,a)$ ➔ the *set* of states you *may end up in* if you execute *a* in state *s*
  - $\gamma(s,a) = \emptyset$ means *a* cannot be executed in *s*
  - $\gamma(s,a) = \{s'\}$ means executing *a* in s leads to s'

The world is always in a given <u>state</u>, which we want to <u>affect</u> → **Another possible state**

A6: Every action results in a <u>discrete</u> state transition

No concept of time

No concept of continuous change (crane swinging from A to B), only:

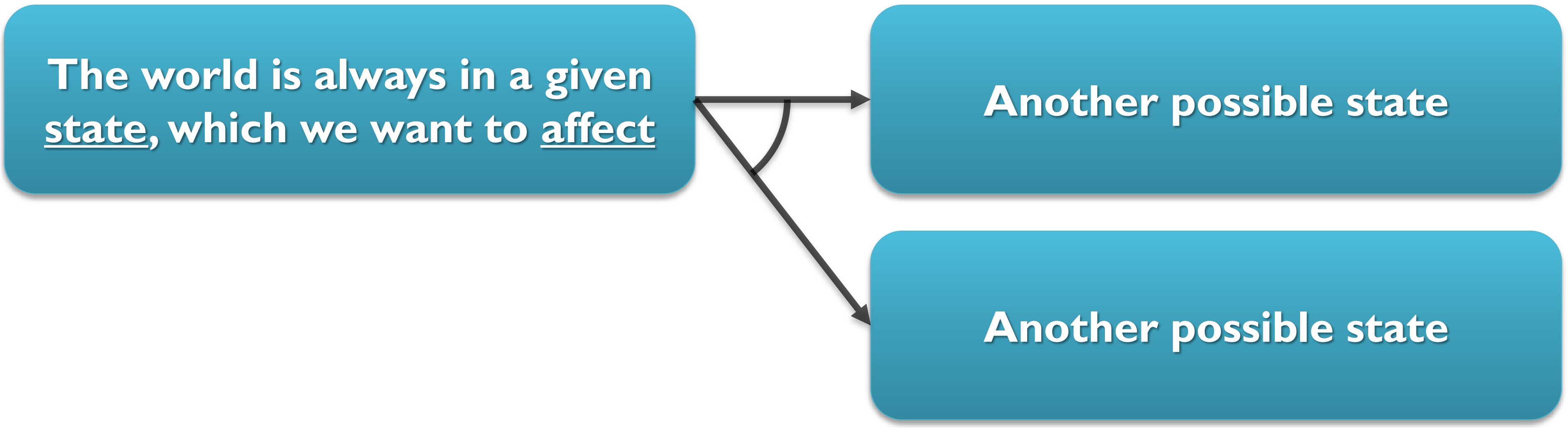
<u>Before</u> *pickup*, the container is on truck *y*;

<u>after</u>, the container is carried by crane z

Many planners do model time in some way ➔ "semi-classical"

Non-determinism?

- $\gamma(s,a) = \{s_1, s_2, \ldots\}$ is possible in some **non-classical planners**



The world is always in a given <u>state</u>, which we want to <u>affect</u>

Another possible state

Another possible state

Non-deterministic actions
Complicated due to explosion of possibilities!

Classical Planning 4

- We must know **when an action is executable** and **what it achieves**
  - $\gamma: S \times A \rightarrow 2^S$ — $\gamma(s,a)$ ➔ returns the set of states you *may end up in*
  - $\gamma(s,a) = \emptyset$ — means *a* cannot be executed in *s*
  - $\gamma(s,a) = \{s'\}$ — means executing *a* in *s* leads to s'
  - $|\gamma(s,a)| > 1$ — impossible in classical planning, due to A2

The world is always in a given state, which we want to affect → **Another possible state**

A2: Deterministic actions
If we know the current state and the action that is executed,
we *know in advance* exactly which state we will end up in

Some planners support non-determinism
Complicated due to explosion of possibilities!

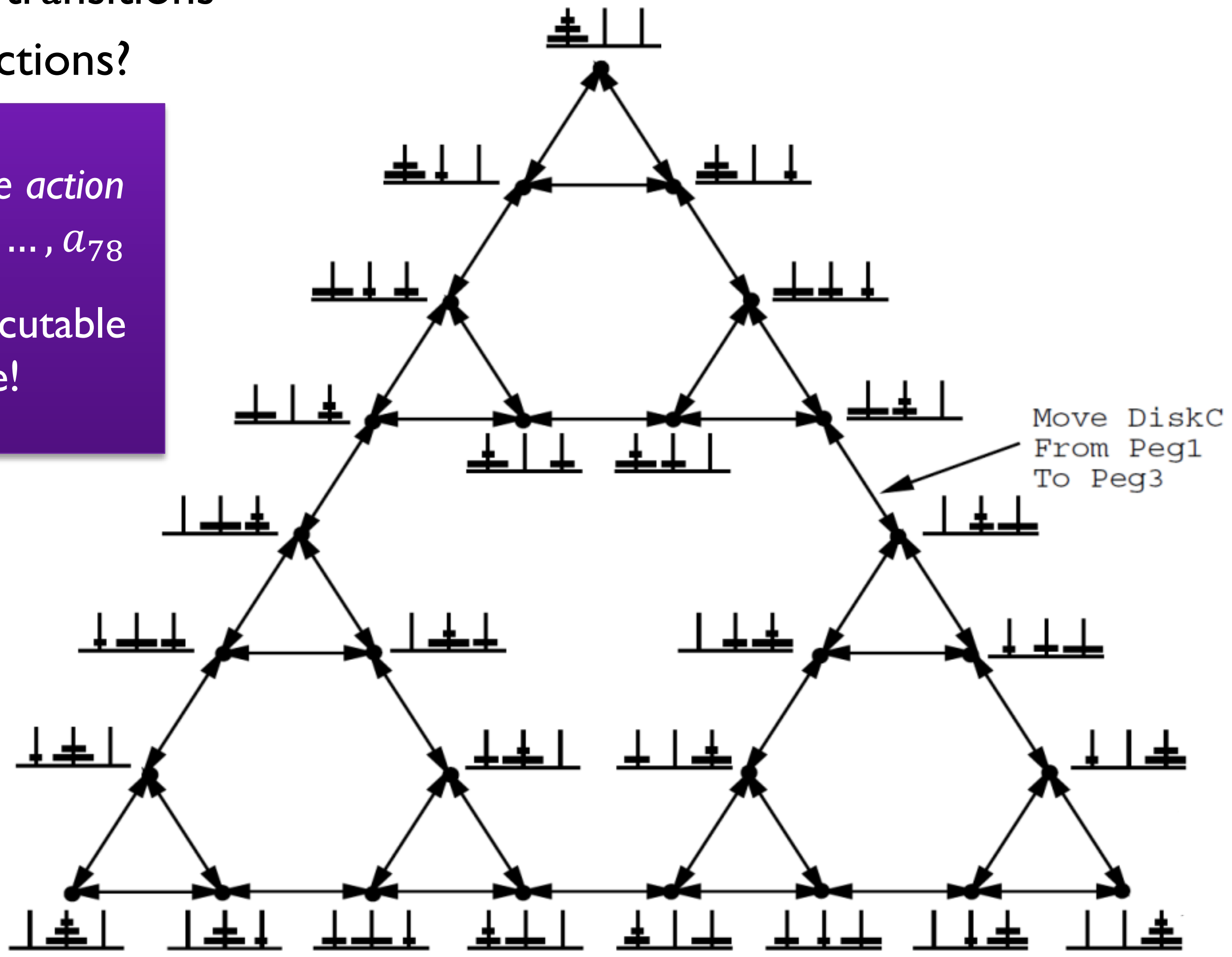
Together: $\Sigma = (S, A, \gamma)$ is a **state transition system (STS)**

Classical Planning 5: STS Example

- **Part** of an **STS** for a Hanoi problem, **illustrated**:
  - 27 states, 78 transitions
  - How many actions?

Can **always** have *one action per transition*: $a_1, a_2, \dots, a_{78}$

Each action only executable in a single state!



Classical Planning 6: STS Example, cont.

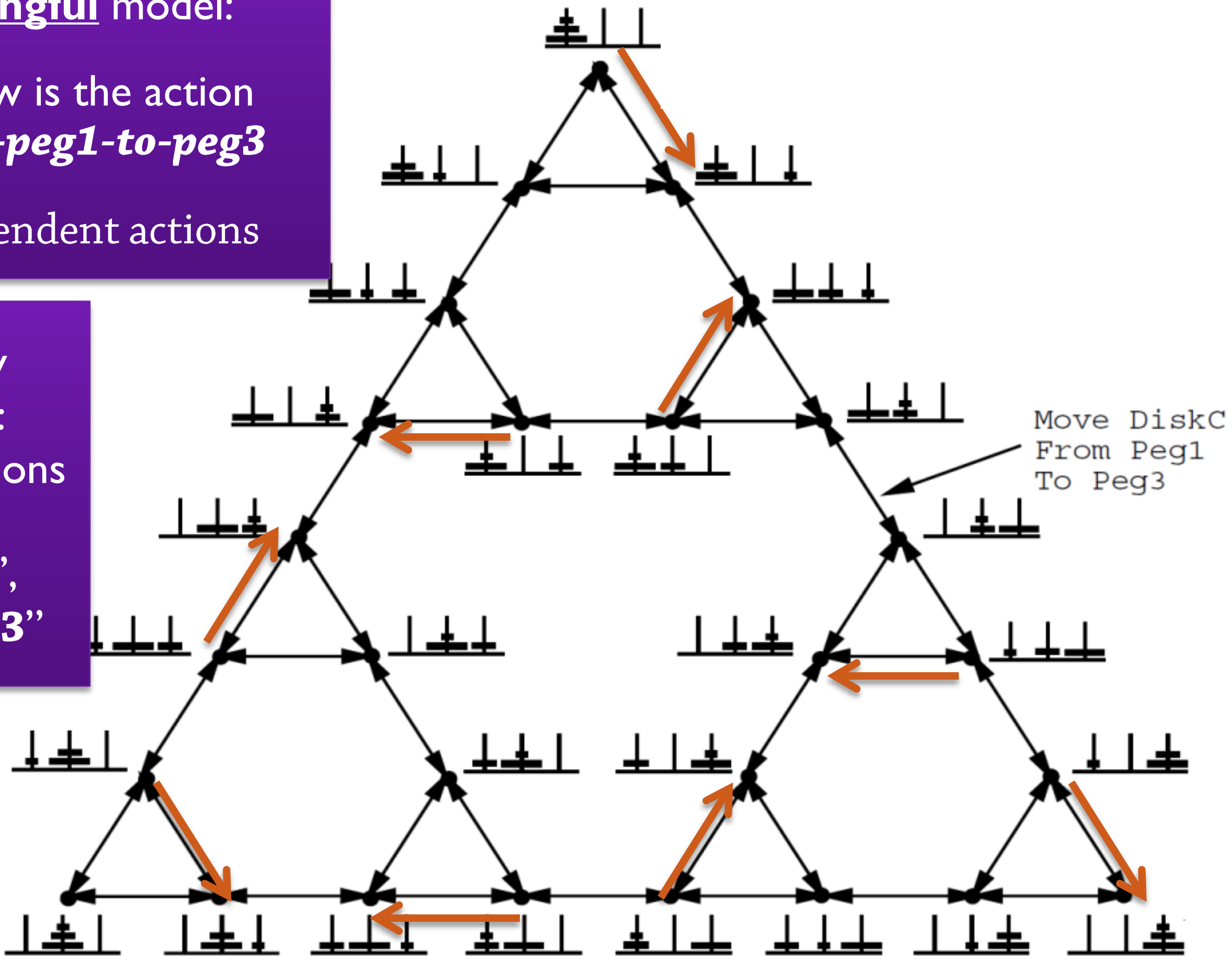
- **Part** of an **STS** for a Hanoi problem, **illustrated**:

Common **meaningful** model:

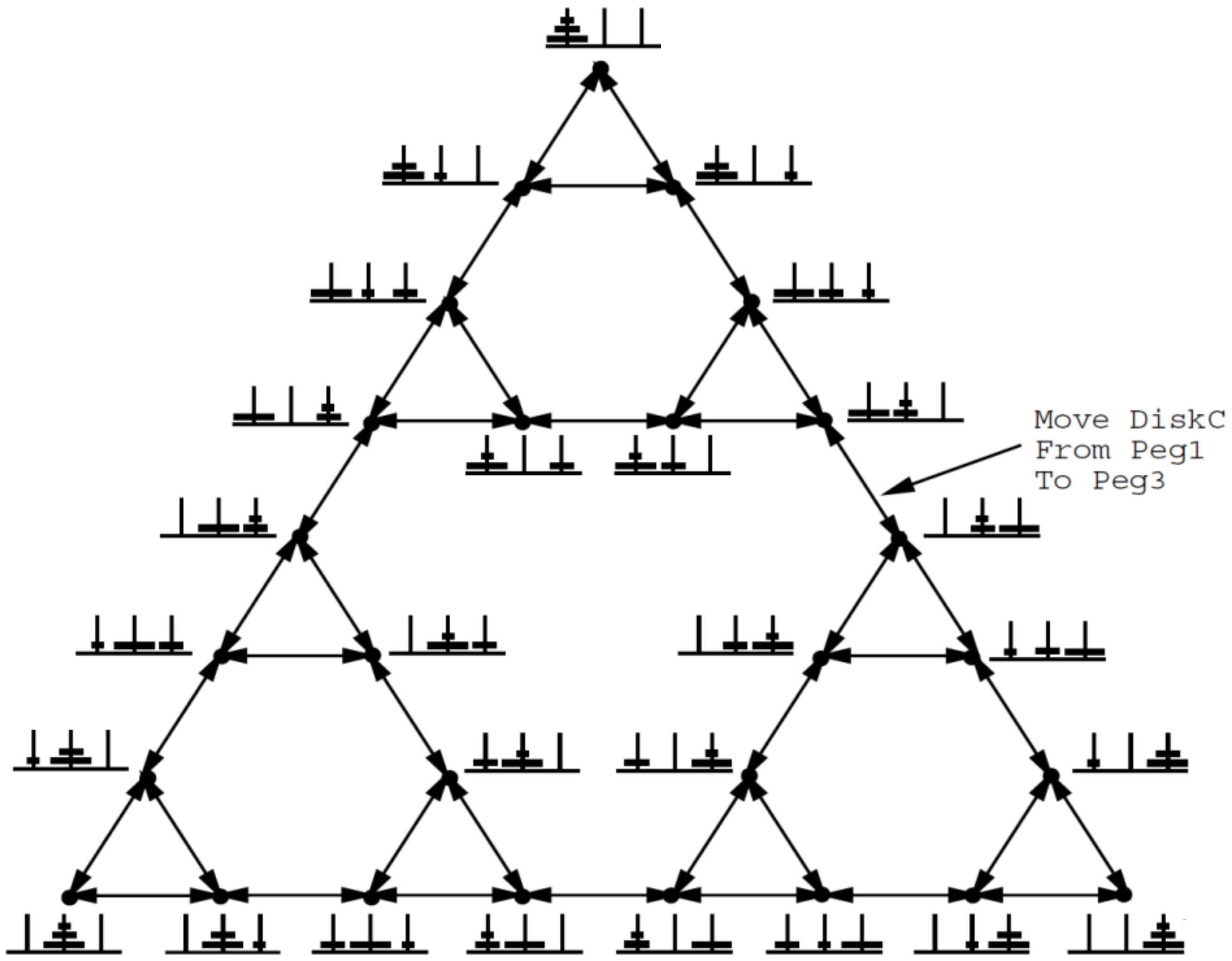
Every orange arrow is the action
move-diskA-from-peg1-to-peg3

➔ 18 context-dependent actions

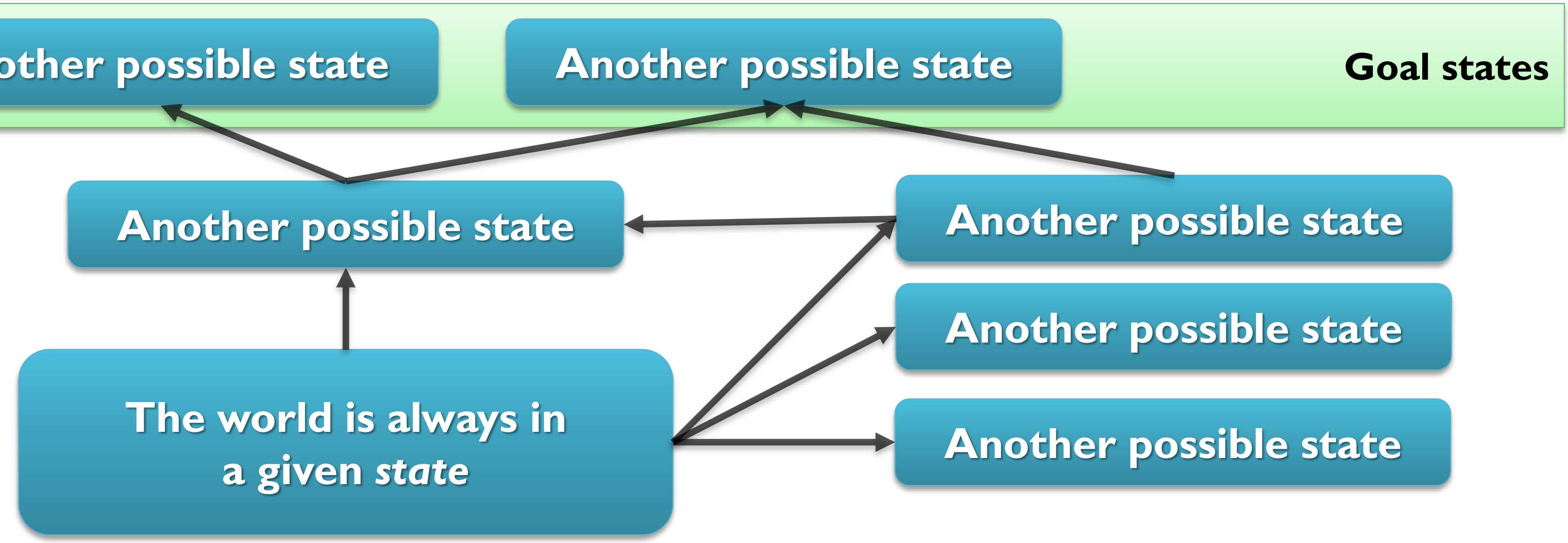
Or we *could* get by using *three* actions:
At most three transitions from any state ➔
Actions "**option1**", "**option2**", "**option3**"



STS: How the world works **<u>in general</u>**
Now: What's the (formal) **<u>problem</u>** to solve?



Problem, part 1: Objectives



A4: Restricted objectives

The *objective* is always to *end up in* a *goal state* $s \in S_g$
(No constraint on cost, time requirements,
states to avoid on the way, …)

Many planners support more complex objectives

Problem, part 2: Initial State

A1: We can always detect the current state

The world is always in a given *state*

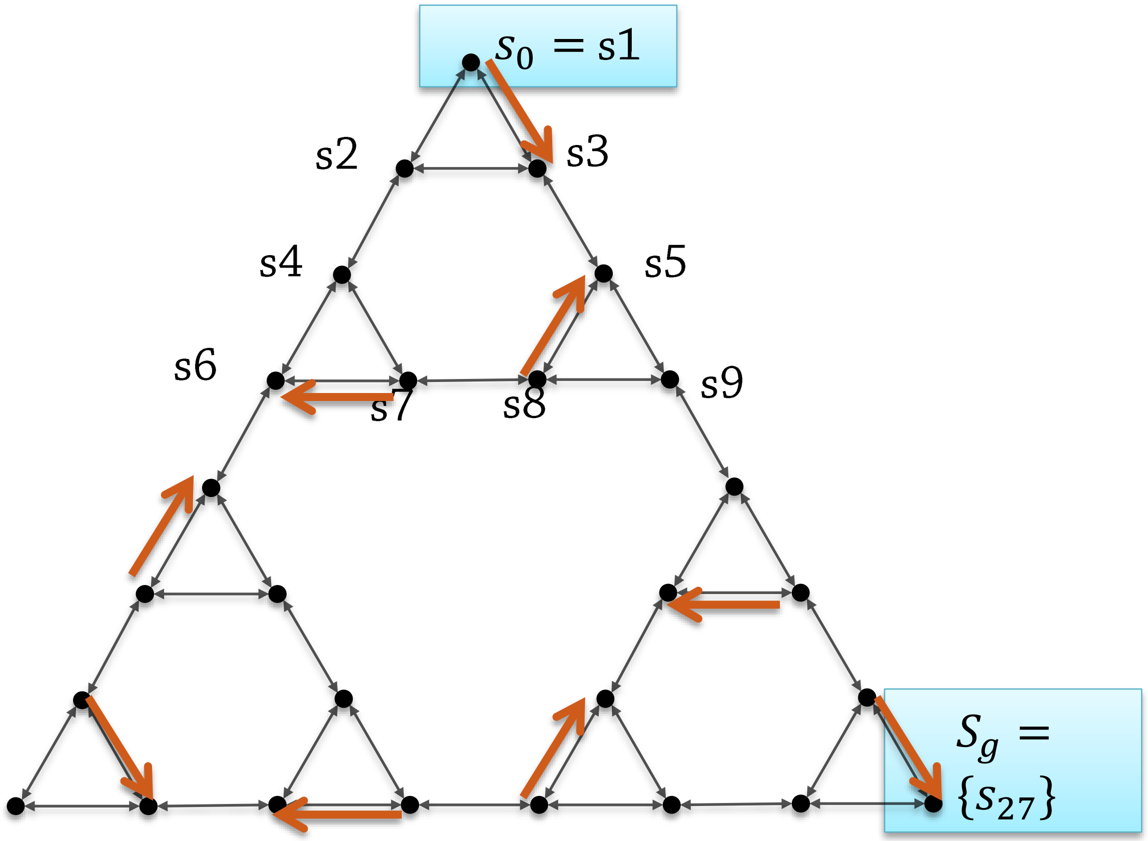
A7: Offline planning
No need to consider changes that may happen *while generating plans.*

We know **now** what the state of the world **will be** when we start executing a plan!

Initial state: s_0

Problem Definition

- Result: A complete **classical planning problem** (Σ, s_0, S_g)



Transition System and Problem

Real World

Abstraction
Approximation
Simplification

State Transition System
$\Sigma = (S, A, \gamma)$

Tells us: **How the world works**
(Only those aspects
that we *need* in our model
in order to solve
interesting problems!)

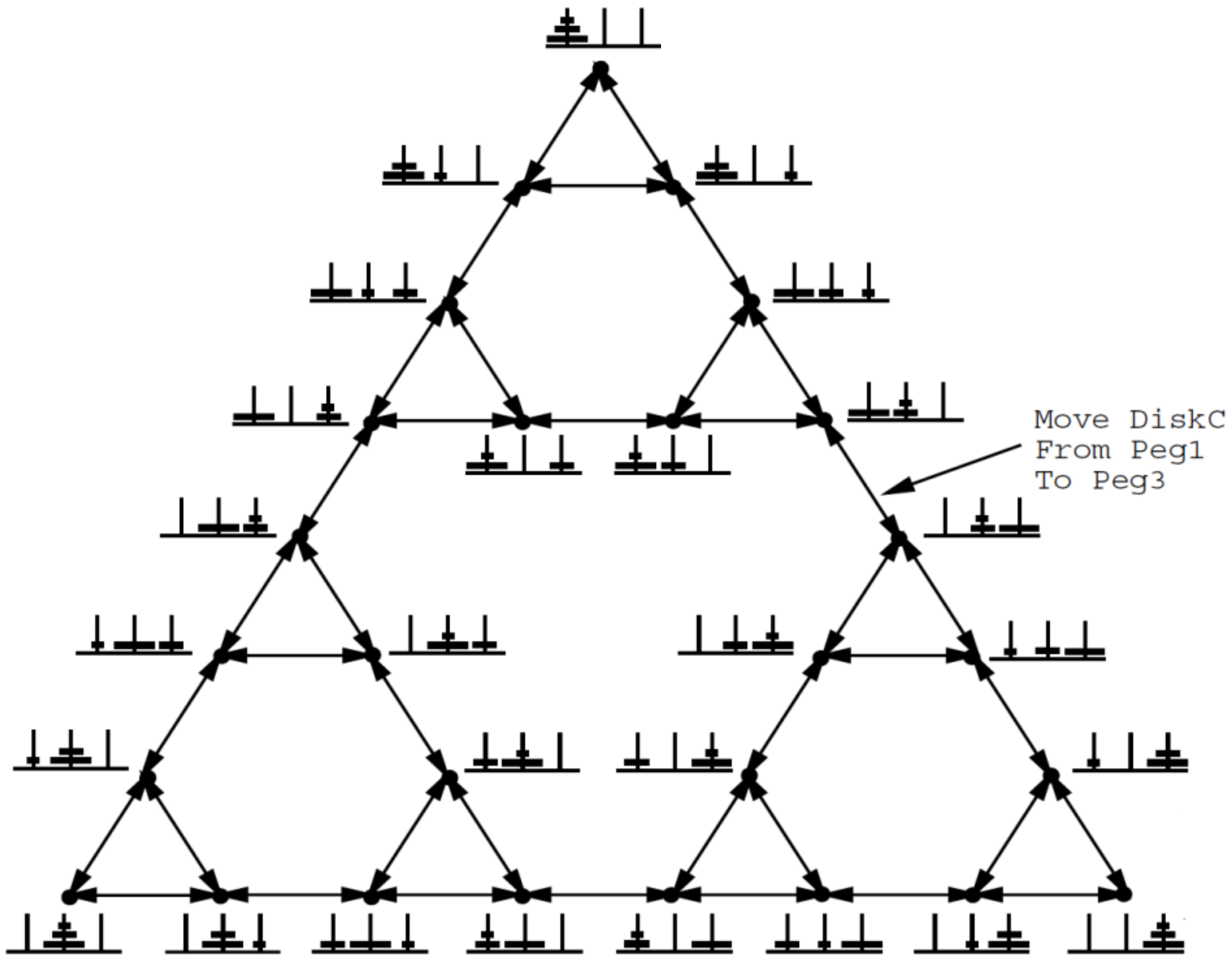
Real World
+ current
problem

Abstraction
Approximation
Simplification

Planning Problem
$\mathcal{P} = (\Sigma, s_0, S_g)$

Tells us:
Which specific problem to solve

And what <u>**is**</u> a solution?



Action Sequences

A5: Sequential execution

A solution never executes two actions concurrently

(Many planners do allow concurrency ➔ "semi-classical")

- **<u>Action sequence</u>**: $\boldsymbol{\sigma = \langle a_1, a_2, \ldots, a_n \rangle}$, where $\{a_1, \ldots, a_n\} \subseteq A$
  - *Sometimes called "plan"*

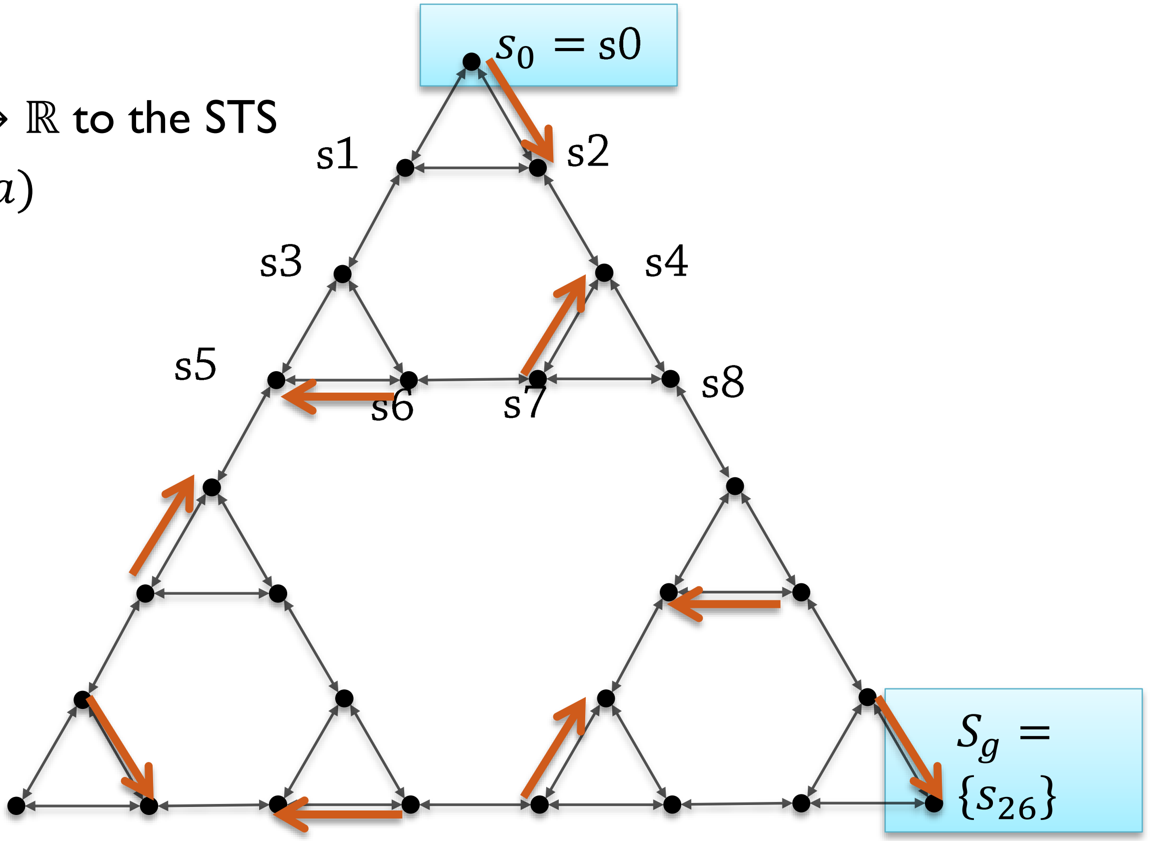
- An action sequence is **<u>executable</u>** in state $s \in S$ if $\exists s_1, \ldots, s_n \in S$ such that:
  - $\gamma(s, a_1) = \{s_1\}$
  - $\gamma(s_1, a_2) = \{s_2\}$
  - $\ldots$
  - $\gamma(s_{n-1}, a_n) = \{s_n\}$
  - *Sometimes called "executable action sequence", "plan", "executable plan", …*

In the exam questions, the terminology will be unambiguous!

Solution

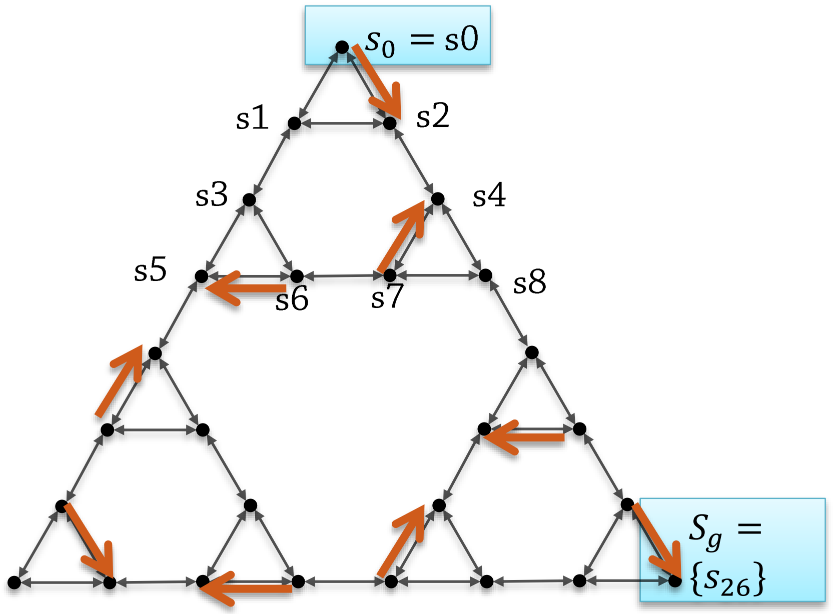
- An action sequence is a **solution** to (Σ, s_0, S_g) if:
  - It is executable in s_0
  - It results in a state $s_n \in S_g$
  - *Sometimes called "plan", "solution plan", …*

- A **good** solution:
  - Add a cost function $c: A \to \mathbb{R}$ to the STS
  - Try to minimize $\sum_{\{a \in \pi\}} c(a)$



Plan Generation (2)

- Is classical planning simply **graph search**?
  - *Can* be, but:
    - **Graphs are enormous**
      Requires advanced heuristics, adapted to planning
      Requires advanced search methods
    - **Alternatives to searching the STS**
      can be used to "indirectly" find paths!
    - Many forms of non-classical planning
      do **not** map into searching an STS



STS 8: Useful?

- Very useful:
  - As a conceptual model, explaining important concepts
  - To analyze expressivity, clarify restrictions
  - To prove properties
- Very useless:
  - As a way of actually **writing down** realistic planning problems (enumerate all possible states?)
  - As an implementation structure for planners
  - → Next time!