AI Planning for Autonomy

3. Introduction to Planning

How to Describe Arbitrary Search Problems

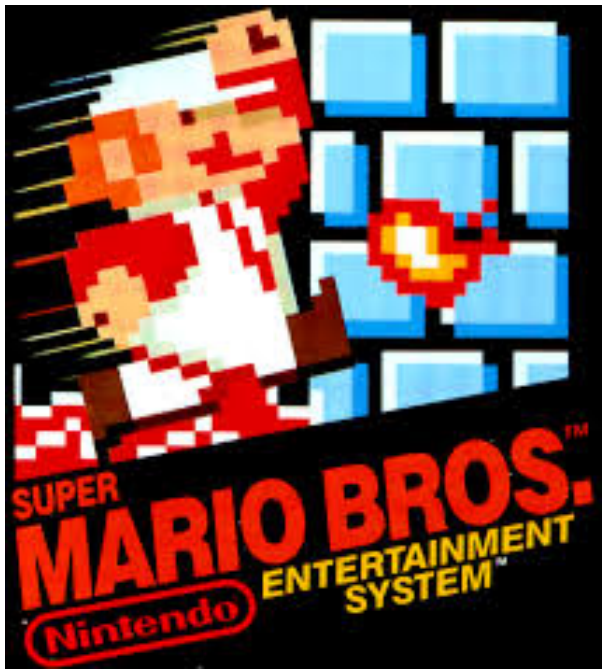
Tim Miller and Nir Lipovetzky

THE UNIVERSITY OF MELBOURNE

Winter Term 2019

Beating Kasparov is great . . .

Beating Kasparov is great . . . but how to play Mario?



- You (and your brother/sister/little nephew) are better than Deep Blue at **everything** - except playing Chess.
- Is that (artificial) 'Intelligence'?

→ How to build machines that automatically solve **new** problems?

Planning: Motivation

How to develop systems or 'agents'
that can make decisions on their own?

Autonomous Behavior in AI

The key problem is to select the action to do next. This is the so-called control problem. Three approaches to this problem:

- **Programming-based:** Specify control by hand
- **Learning-based:** Learn control from experience
- **Model-based:** Specify problem by hand, derive control automatically

→ Approaches not orthogonal; successes and limitations in each . . .

→ Different models yield different types of controllers . . .

Programming-Based Approach

→ Control specified by programmer, e.g.:

- If Mario finds no danger, then run...
- If danger appears and Mario is big, jump and kill ...
- ...

- Advantage: domain-knowledge easy to express
- Disadvantage: cannot deal with situations not anticipated by programmer

Learning-Based Approach

→ Learns a controller from experience or through simulation:

- **Unsupervised** (Reinforcement Learning):
  - penalize Mario each time that 'dies'
  - reward agent each time oponent 'dies' and level is finished, . . .
- **Supervised** (Classification)
  - learn to classify actions into good or bad from info provided by teacher
- **Evolutionary**:
  - from pool of possible controllers: try them out, select the ones that do best, and mutate and recombine for a number of iterations, keeping best

- Advantage: does not require much knowledge in principle
- Disadvantage: in practice, hard to know which features to learn, and is slow

General Problem Solving

Ambition: Write one program that can solve all problems.

$\rightarrow$ Write $X \in \{algorithms\}$: for all $Y \in \{`problems'\}$: X‘solves’Y

$\rightarrow$ What is a ‘problem’? What does it mean to ‘solve’ it?

Ambition 2.0: Write one program that can solve a large class of problems

Ambition 3.0: Write one program that can solve a large class of problems effectively

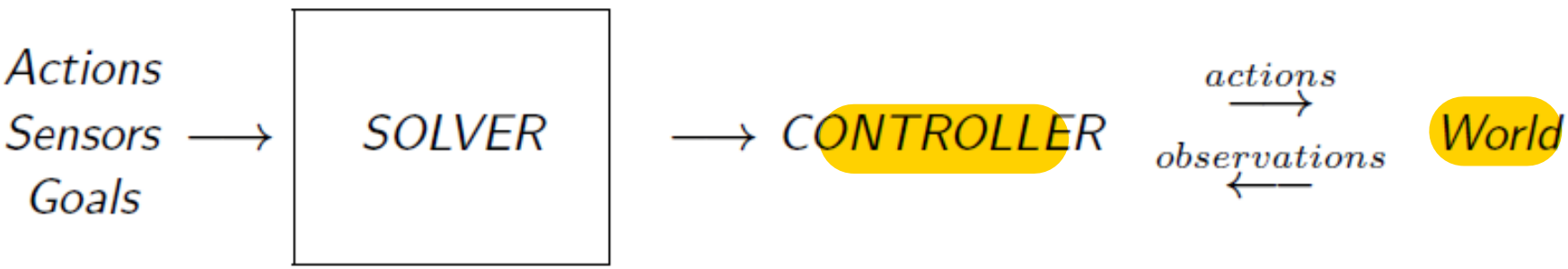
(some new problem) $\rightsquigarrow$ (describe problem $\rightarrow$ use off-the-shelf solver) $\rightsquigarrow$ (solution competitive with a human-made specialized program)

$\rightarrow$ Beat humans at coming up with clever solution methods!
(Link: GPS started on 1959)

Model-Based Approach / General Problem Solving

→ specify model for problem: actions, initial situation, goals, and sensors

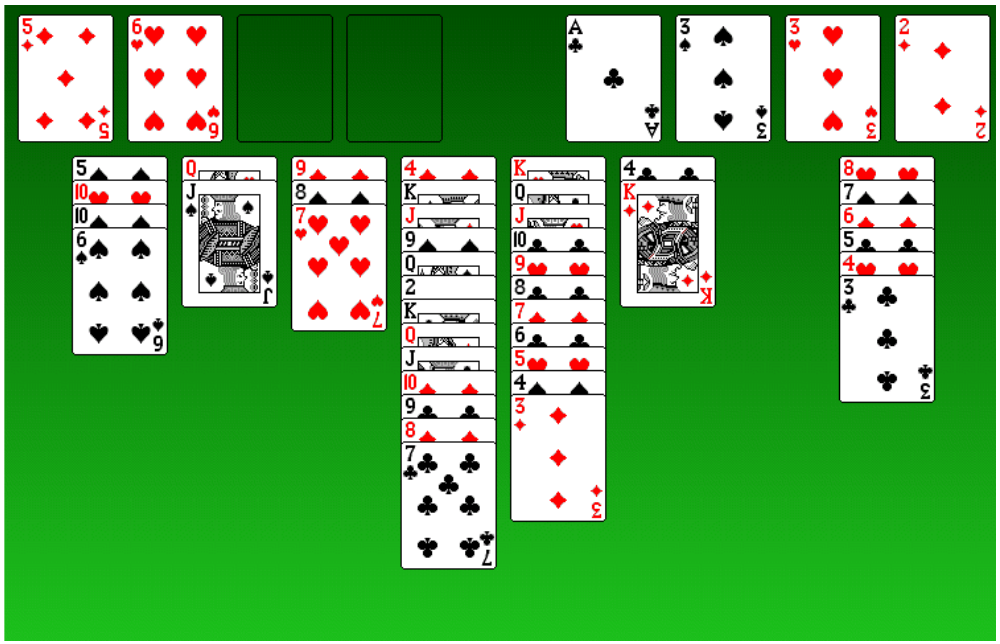
→ let a solver compute controller automatically



→ Advantage:

- Powerful: In some applications generality is absolutely necessary
- Quick: Rapid prototyping. 10s lines of problem description vs. 1000s lines of C++ code. (Language generation!)
- Flexible & Clear: Adapt/maintain the description.

Example: Classical Search Problem



- 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.

Basic State Model: Classical Planning

Ambition:

Write one program that can solve all classical search problems.

State Model:

- finite and discrete state space S
- a known initial state $s_0 \in S$
- a set $S_G \subseteq S$ of goal states
- actions $A(s) \subseteq A$ applicable in each $s \in S$
- a deterministic transition function $s' = f(a, s)$ for $a \in A(s)$
- positive action costs $c(a, s)$

$\rightarrow$ A **solution** is a sequence of applicable actions that maps s_0 into S_G, and it is **optimal** if it minimizes **sum of action costs** (e.g., # of steps)

$\rightarrow$ Different **models** and **controllers** obtained by relaxing assumptions in **blue** ...

Uncertainty but No Feedback: Conformant Planning

- finite and discrete state space S
- a set of possible initial state $S_0 \in S$
- a set $S_G \subseteq S$ of goal states
- actions $A(s) \subseteq A$ applicable in each $s \in S$
- a non-deterministic transition function $F(a, s) \subseteq S$ for $a \in A(s)$
- uniform action costs $c(a, s)$

→ A **solution** is still an **action sequence** but must achieve the goal for **any possible initial state and transition**

→ More complex than **classical planning**, verifying that a plan is **conformant** intractable in the worst case; but special case of **planning with partial observability**

Planning with Markov Decision Processes

MDPs are **fully observable, probabilistic** state models:

- a state space S
- initial state $s_0 \in S$
- a set $G \subseteq S$ of goal states
- actions $A(s) \subseteq A$ applicable in each state $s \in S$
- transition probabilities $P_a(s'|s)$ for $s \in S$ and $a \in A(s)$
- action costs $c(a, s) > 0$

→ **Solutions** are **functions (policies)** mapping states into actions

→ **Optimal** solutions minimize **expected cost** to goal

Partially Observable MDPs (POMDPs)

POMDPs are **partially observable, probabilistic** state models:

- states $s \in S$
- actions $A(s) \subseteq A$
- transition probabilities $P_a(s'|s)$ for $s \in S$ and $a \in A(s)$
- initial belief state b_0
- final belief state b_f
- sensor model given by probabilities $P_a(o|s)$, $o \in Obs$

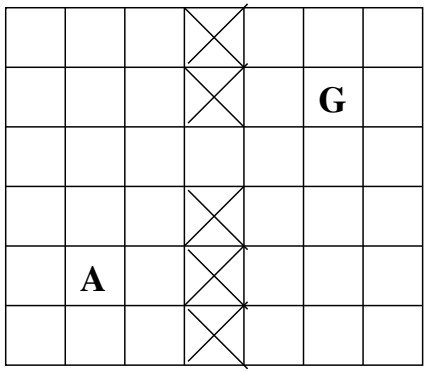
→ **Belief states** are probability distributions over S

→ **Solutions** are policies that map belief states into actions

→ **Optimal** policies minimize **expected** cost to go from b_0 to G

Example

Agent **A** must reach **G**, moving one cell at a time in **known** map



- If actions deterministic and initial location known, planning problem is **classical**
- If actions stochastic and location observable, problem is an **MDP**
- If actions stochastic and location partially observable, problem is a **POMDP**

Different combinations of uncertainty and feedback: three problems, three models

Models, Languages, and Solvers

- A planner is a solver over a class of models; it takes a model description, and computes the corresponding controller

$$\textit{Model} \Longrightarrow \boxed{\textit{Planner}} \Longrightarrow \textit{Controller}$$

- Many models, many solution forms: uncertainty, feedback, costs, ...
- Models described in suitable planning languages (Strips, PDDL, PPDDL, ...) where states represent interpretations over the language.

A Basic Language for Classical Planning: Strips

- A **problem** in STRIPS is a tuple $P = \langle F, O, I, G \rangle$:
  - F stands for set of all atoms (boolean vars)
  - O stands for set of all operators (actions)
  - $I \subseteq F$ stands for initial situation
  - $G \subseteq F$ stands for goal situation
- Operators $o \in O$ **represented** by
  - the Add list $Add(o) \subseteq F$
  - the Delete list $Del(o) \subseteq F$
  - the Precondition list $Pre(o) \subseteq F$

From Language to Models (STRIPS Semantics)

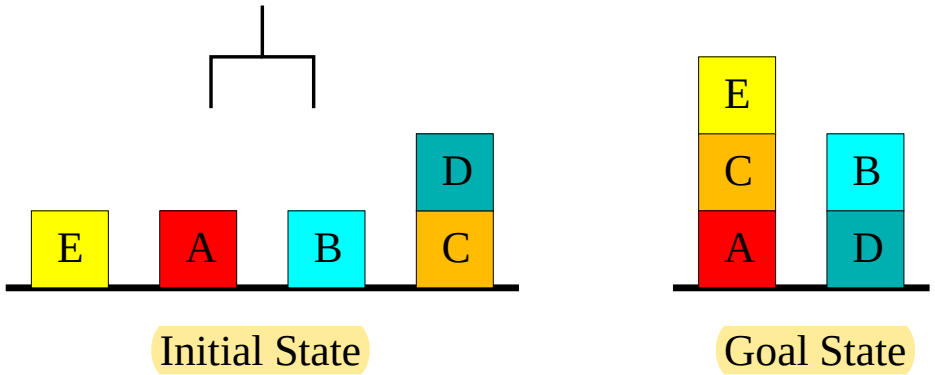
A STRIPS problem $P = \langle F, O, I, G \rangle$ determines **state model** $\mathcal{S}(P)$ where

- the states $s \in S$ are collections of atoms from F. $S = 2^F$
- the initial state s_0 is I
- the goal states s are such that $G \subseteq s$
- the actions a in $A(s)$ are ops in O s.t. $Prec(a) \subseteq s$
- the next state is $s' = s - Del(a) + Add(a)$
- action costs $c(a, s)$ are all 1

$\rightarrow$ (Optimal) **Solution** of P is (optimal) **solution** of $\mathcal{S}(P)$

$\rightarrow$ Slight language extensions often convenient: **negation**, **conditional effects**, **non-boolean variables**; some required for describing richer models (costs, probabilities, ...).

(Oh no it's) The Blocksworld



Initial State Goal State

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

PDDL Quick Facts

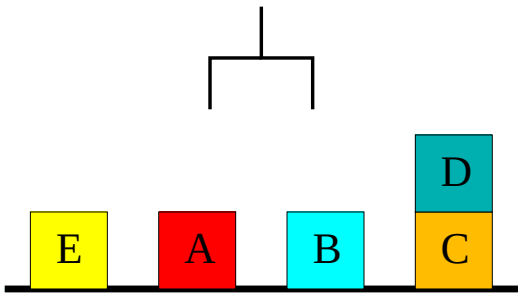
PDDL is not a propositional language:

- Representation is lifted, using object variables to be instantiated from a finite set of objects. (Similar to predicate logic)
- Action schemas parameterized by objects.
- Predicates to be instantiated with objects.

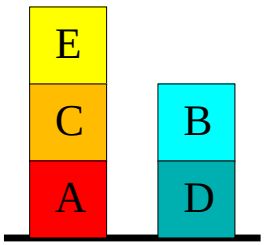
A PDDL planning task comes in two pieces:

- The domain file and the problem file.
- The problem file gives the objects, the initial state, and the goal state.
- The domain file gives the predicates and the operators; each benchmark domain has *one* domain file.

The Blocksworld in PDDL: Domain File



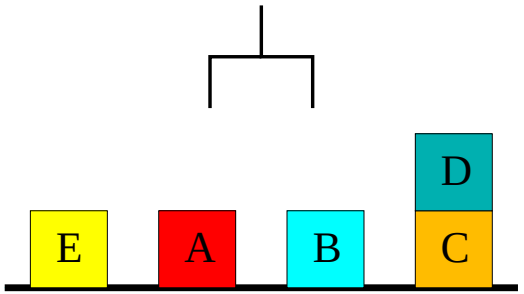
Initial State



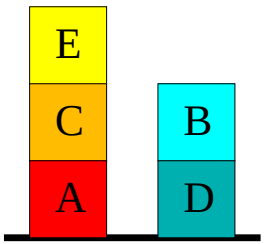
Goal State

```
(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)))
)
...
```

The Blocksworld in PDDL: Problem File



Initial State



Goal State

```
(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))))
```

Example: Logistics in Strips PDDL

```
(define (domain logistics)
  (:requirements :strips :typing :equality)
  (:types airport - location  truck airplane - vehicle  vehicle packet -
  (:predicates (loc-at ?x - location ?y - city) (at ?x - thing ?y - locat
  (:action load
    :parameters (?x - packet ?y - vehicle ?z - location)
    :precondition (and (at ?x ?z) (at ?y ?z))
    :effect (and (not (at ?x ?z)) (in ?x ?y)))
  (:action unload ..)
  (:action drive
    :parameters (?x - truck ?y - location ?z - location ?c - city)
    :precondition (and (loc-at ?z ?c) (loc-at ?y ?c) (not (= ?z ?y)) (at
    :effect (and (not (at ?x ?z)) (at ?x ?y)))
...
(define (problem log3_2)
  (:domain logistics)
  (:objects packet1 packet2 - packet  truck1 truck2 truck3 - truck  airpl
  (:init (at packet1 office1) (at packet2 office3) ...)
  (:goal (and (at packet1 office2) (at packet2 office2))))
```

Algorithmic Problems in Planning

Satisficing Planning

Input: A planning task P.
Output: A plan for P, or 'unsolvable' if no plan for P exists.

Optimal Planning

Input: A planning task P.
Output: An optimal plan for P, or 'unsolvable' if no plan for P exists.

$\rightarrow$ The techniques successful for either one of these are almost disjoint!

$\rightarrow$ Satisficing planning is much more effective in practice

$\rightarrow$ Programs solving these problems are called (optimal) planners, planning systems, or planning tools.

Decision Problems in Planning

Definition (PlanEx). By PlanEx, we denote the problem of deciding, given a planning task P, whether or not there exists a plan for P.

→ Corresponds to satisficing planning.

Definition (PlanLen). By PlanLen, we denote the problem of deciding, given a planning task P and an integer B, whether or not there exists a plan for P of length at most B.

→ Corresponds to optimal planning.

Reminder (?): **NP** and **PSPACE**

Def Turing machine: Works on a tape consisting of tape cells, across which its R/W head moves. The machine has internal states. There are transition rules specifying, given the current cell content and internal state, what the subsequent internal state will be, and whether the R/W head moves left or right or remains where it is. Some internal states are accepting ('yes'; else 'no').

Def NP: Decision problems for which there exists a non-deterministic Turing machine that runs in time polynomial in the size of its input. Accepts if at least one of the possible runs accepts.

Def PSPACE: Decision problems for which there exists a deterministic Turing machine that runs in space polynomial in the size of its input.

Relation: Non-deterministic polynomial space can be simulated in deterministic polynomial space. Thus **PSPACE = NPSPACE**, and hence (trivially) **NP** *subset* **PSPACE**.

→ For comprehensive details, please see a text book. My personal favorite is [Garey and Johnson (1979)].

Computational Complexity of PlanEx and PlanLen

Theorem. PlanEx and PlanLen is **PSPACE**-complete.

→ 'At least as hard as any other problem contained in **PSPACE**.'

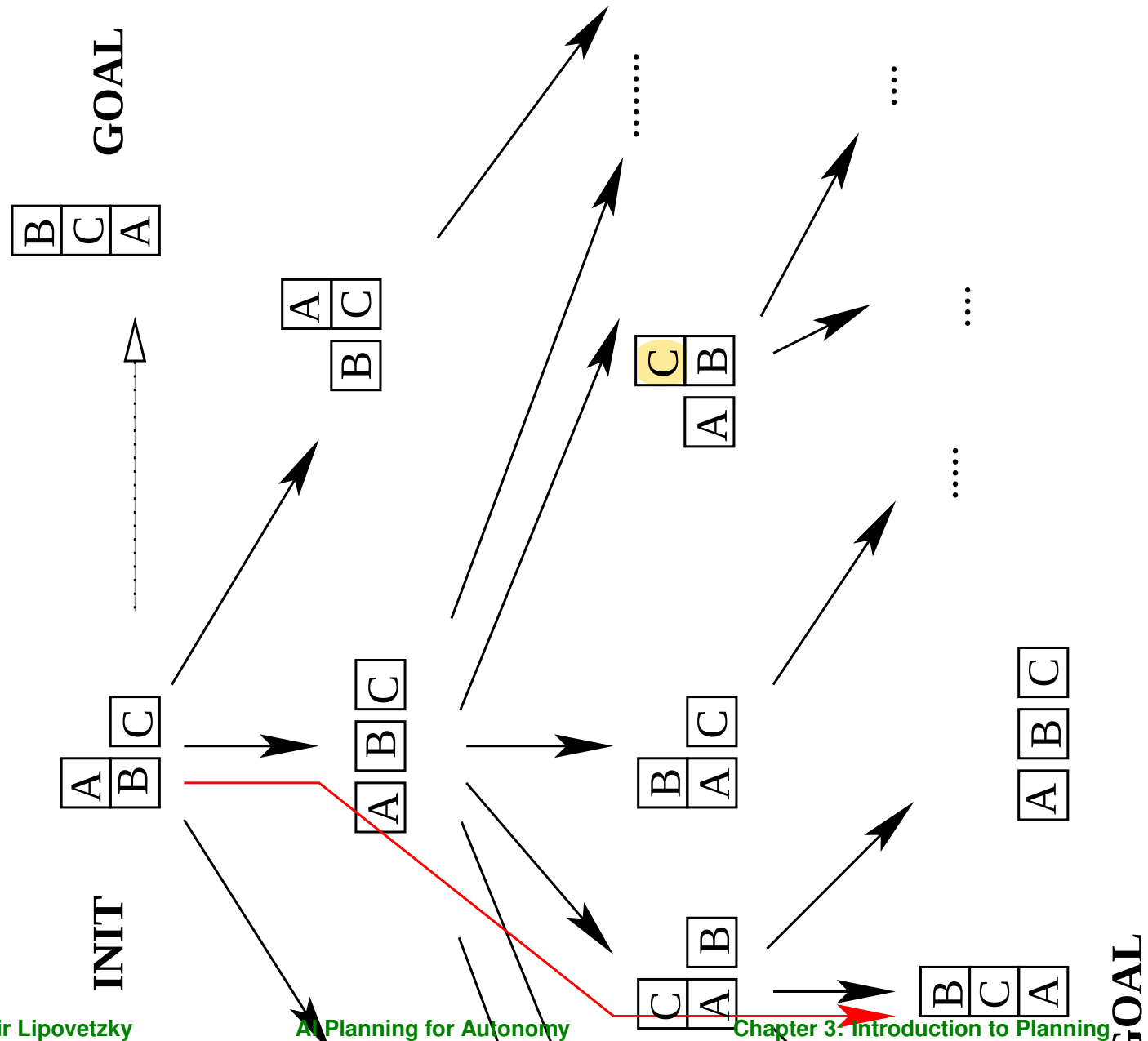
→ Details: [Bylander (1994)]

Domain-Specific PlanEx vs. PlanLen . . .

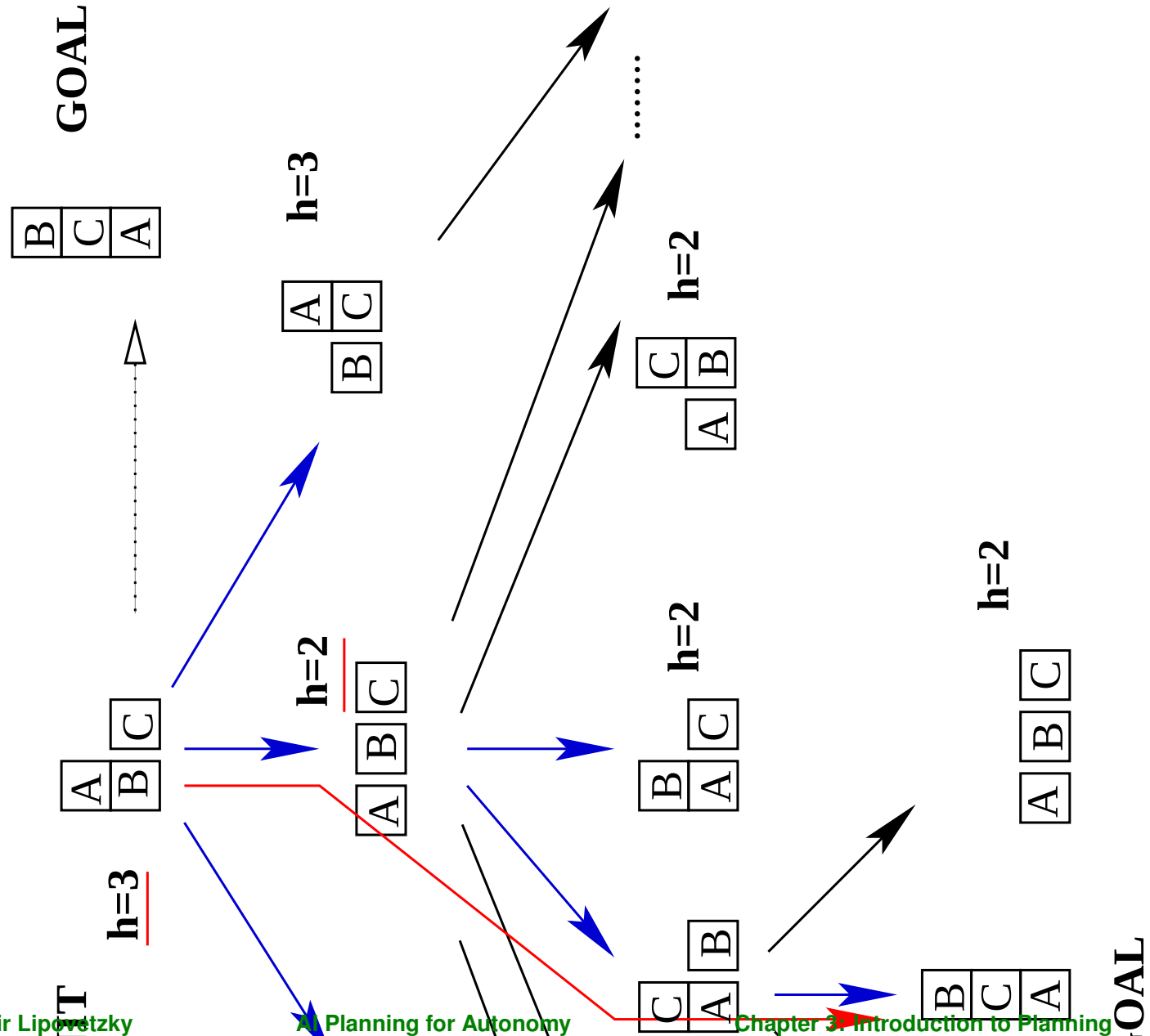
- In general, both have the same complexity.
- Within particular applications, bounded length plan existence is often harder than plan existence.
- This happens in many IPC benchmark domains: PlanLen is **NP**-complete while PlanEx is in **P**.
  - For example: Blocksworld and Logistics.

→ In practice, optimal planning is (almost) never 'easy'

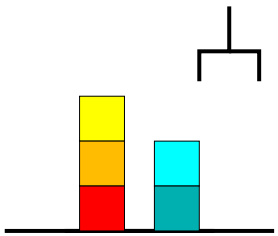
The Blocksworld is Hard?



The Blocksworld is Hard!



So, Why All the Fuss? Example Blocksworld



- n blocks, 1 hand.
- A single action either takes a block with the hand or puts a block we're holding onto some other block/the table.

blocks	states	blocks	states
1	1	9	4596553
2	3	10	58941091
3	13	11	824073141
4	73	12	12470162233
5	501	13	202976401213
6	4051	14	3535017524403
7	37633	15	65573803186921
8	394353	16	1290434218669921

$\rightarrow$ State spaces may be huge. In particular, the state space is typically exponentially large in the size of its specification via the problem Π (up next).

$\rightarrow$ In other words: Search problems typically are computationallly hard (e.g., optimal Blocksworld solving is **NP**-complete).

Computation: how to solve Strips planning problems?

Key issue: exploit two roles of **language**:

- specifcation: concise model description
- computation: reveal useful heuristic information (structure)

Two traditional approaches: search vs. decomposition

- explicit search of the state model $S(P)$ direct but not effective til recently
- near decomposition of the planning problem thought a better idea

Computational Approaches to Classical Planning

- **General Problem Solver (GPS) and Strips** (50's-70's): mean-ends analysis, decomposition, regression, ...
- **Partial Order (POCL) Planning** (80's): work on any open subgoal, resolve threats; UCPOP 1992
- **Graphplan** (1995 – 2000): build graph containing all possible **parallel** plans up to certain length; then extract plan by searching the graph backward from Goal
- **SATPlan** (1996 – ...): map planning problem given horizon into SAT problem; use state-of-the-art SAT solver
- **Heuristic Search Planning** (1996 – ...): search state space $\mathcal{S}(P)$ with heuristic function h extracted from problem P
- **Model Checking Planning** (1998 – ...): search state space $\mathcal{S}(P)$ with 'symbolic' Breadth first search where sets of states represented by formulas implemented by BDDs ...

State of the Art in Classical Planning

- significant **progress** since Graphplan
- **empirical methodology**
  - standard PDDL language
  - planners and benchmarks available; competitions
  - focus on performance and scalability
- **large problems solved** (non-optimally)
- different **formulations** and **ideas**
  1. Planning as **Heuristic Search**
  2. Planning as **SAT**
  3. **Other:** Local Search (LPG), Monte-Carlo Search (Arvand), ...

I'll focus on **1** mainly, and partially on **2**

The International Planning Competition (IPC)

Competition?

'Run competing planners on a set of benchmarks devised by the IPC organizers. Give awards to the most effective planners.'

- 1998, 2000, 2002, 2004, 2006, 2008, 2011, 2014
- PDDL [McDermott and others (1998); Fox and Long (2003); Hoffmann and Edelkamp (2005)]
- $\approx$ 40 domains, $\gg$ 1000 instances, 74 (!!) planners in 2011
- Optimal track vs. satisficing track
- Various others: uncertainty, learning, . . .

http://ipc.icaps-conference.org/

... Winners

- IPC 2000: Winner FF, heuristic search (HS), IPC 2002: Winner LPG, HS
- IPC 2004: Winner satisficing SGPlan, HS; optimal SATPLAN, compilation to SAT
- IPC 2006: Winner satisficing SGPlan, HS; optimal SATPLAN, compilation to SAT
- IPC 2008: Winner satisficing LAMA, HS; optimal Gamer, symbolic search
- IPC 2011: Winner satisficing LAMA, HS; optimal Fast-Downward, HS
- IPC 2014: Winner satisficing IBACOP, HS Portfolio; optimal SymbA*, symbolic search
- IPC 2018: Winner satisficing FD/BFWS-LAPKT, HS Portfolio/Width-Based planning; optimal Delfi, HS portfolio

→ For the rest of this chapter, we focus on planning as heuristic search
→ This is a VERY short summary of the history of the IPC! There are many different categories, and many different awards

Disclaimer on IPC

Question

If planners x, y both compete in IPC'YY, and x wins, is x 'better than' y?

(A): Yes. (B): No.

$\rightarrow$ Yes, but only on the IPC'YY benchmarks, and only according to the criteria used for determining a 'winner'! On other domains and/or according to other criteria, you may well be better off with the 'looser'.

$\rightarrow$ It's complicated, over-simplification is dangerous. (But, of course, nevertheless is being done all the time).

Summary

- General problem solving attempts to develop solvers that perform well across a large class of problems.
- Planning, as considered here, is a form of general problem solving dedicated to the class of classical search problems. (Actually, we also address inaccessible, stochastic, dynamic, continuous, and multi-agent settings.)
- Classical search problems require to find a path of actions leading from an initial state to a goal state.
- They assume a single-agent, fully-observable, deterministic, static environment. Despite this, they are ubiquitous in practice.
- Heuristic search planning has dominated the International Planning Competition (IPC). We focus on it here.
- 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.
- Plan existence (bounded or not) is **PSPACE**-complete to decide for STRIPS.
- PDDL is the de-facto standard language for describing planning problems.

Reading, ctd.

- *Everything You Always Wanted to Know About Planning (But Were Afraid to Ask)* [*Joerg Hoffmann, 2011*]

  Available at:

  `http://fai.cs.uni-saarland.de/hoffmann/papers/ki11.pdf`

  Content: Joerg personal perspective on planning. Very modern indeed. Excerpt from the abstract:

  The area has long had an affinity towards playful illustrative examples, imprinting it on the mind of many a student as an area concerned with the rearrangement of blocks, and with the order in which to put on socks and shoes (not to mention the disposal of bombs in toilets). Working on the assumption that this "student" is you – the readers in earlier stages of their careers – I herein aim to answer three questions that you surely desired to ask back then already:

  What is it good for? Does it work? Is it interesting to do research in?

Extra material

Introduction to STRIPS, from simple games to StarCraft:
`http://www.primaryobjects.com/2015/11/06/artificial-intelligence-planning-with-strips-a-gentle-introduction/`

Online Editor to model in PDDL:
`http://editor.planning.domains`