Lecture 10: Logical Agents and Planning

Viliam Lisý & Branislav Bošanský

Artificial Intelligence Center
Department of Computer Science, Faculty of Electrical Eng.
Czech Technical University in Prague

viliam.lisy@fel.cvut.cz

April, 2021

Plan of today's lecture

1. Logic in AI in the past and now
2. Logical problem representations
3. Situation calculus
4. Intelligent planning

Acknowledgements

Slides are heavily based on J. Klema's slides. For more details on logical agents see his video from the last year.

Logics in AI

There has been a big hype of logical agents in 60s and 70s.

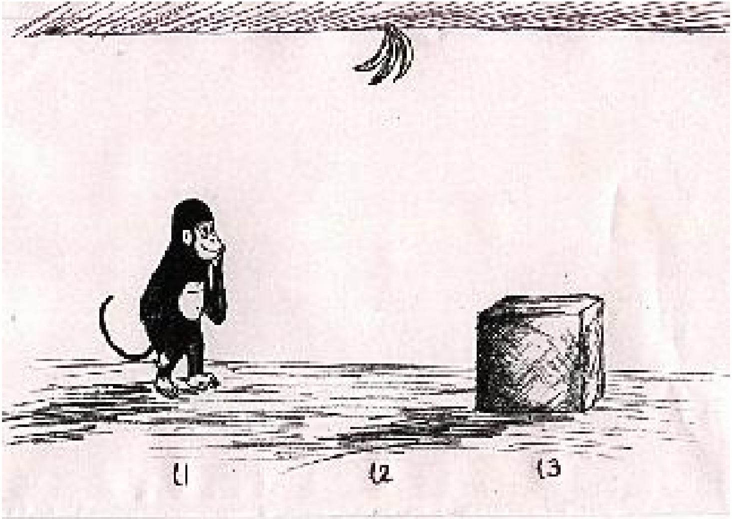
+ It can represent knowledge about the world
+ It can represent intelligent reasoning
- It is not very convenient for working with uncertainty
- It is usually extremely computationally expensive

  (expresivity vs. completeness vs. effectivity)

Logic in AI 2020s

- Interpretable AI
- Relational ML/RL
- Theorem proving
- Model checking
- Knowledge graphs
- Automated planning

Motivation example – monkey and banana



Vladimir Lifschitz: Planning course, The University of Texas at Austin.

Motivation example – monkey and banana

Problem description

- a monkey is in a room, a banana hangs from the ceiling,
- the banana is beyond the monkey's reach,
- the monkey is able to walk, move and climb objects, grasp banana,
- the room is just the right height so that the monkey can move a box, climb it and grasp the banana,
- the goal is to generate this plan (sequence of actions) automatically.

Key characteristics

- a deterministic task
- a general description available
  - all the necessary knowledge is provided
  - we need to **represent it** in some **language**
  - and perform certain **reasoning / inference**
- a planning task

Language $\rightarrow$ First Order (Predicate) Logic (FOL)

Remember B0B01LGR: Logic and Graphs

Jazyk

Jazyk predikátové logiky obsahuje tyto symboly:

1. logické symboly
   - proměnné; Var je množina všech proměnných
   - logické spojky: $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$, popř. též $\mathrm{tt}, \mathrm{ff}, |, \downarrow, \oplus$
   - kvantifikátory $\forall$ (obecný) a $\exists$ (existenční)
   - symbol rovnosti: $=$
2. speciální symboly
   - predikátové, kde každý má svou aritu $n \geq 0$;
     Pred je množina predikátových symbolů
   - funkční, kde každý má svou aritu $n > 0$;
     Func je množina funkčních symbolů
   - konstantní; Kons je množina konstantních symbolů
3. pomocné symboly, jako jsou závorky (,) a čárka ,

The following slides would, in principle, work with stronger logic!
Modal Logic, epistemic logic, temporal logic, ATL

Planning problem representation in FOL

Situation calculus is one way to represent changing wrold in FOL

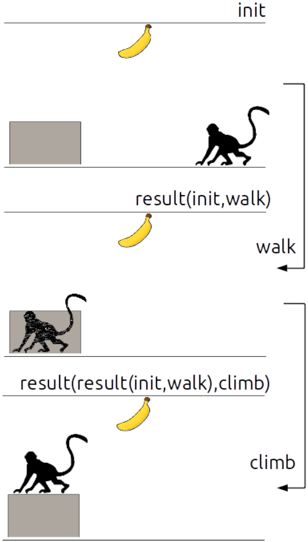
- facts hold in particular situations ($\approx$ world state histories)
- predicates either rigid (eternal) or fluent (changing)
- fluent predicates include a situation argument
  e.g., $agent(monkey, at\_ban, now)$, term now denotes a situation
- rigid predicates hold regardless of a situation
  e.g., $walks(monkey)$, $moveable(box)$
- situations are connected by the *result* function
  if s is a situation than $result(s, a)$ is also a situation

The monkey problem state can be represented using two predicates

- $agent(agent\ name, agent\ position, stands\ on, situation)$
- $object(object\ name, object\ position, who\ stands, situation)$

Keeping track of evolving situations

agent(*agent name*, *agent position*, *stands on*, *situation*)
object(*object name*, *object position*, *who stands*, *situation*)



agent(monkey, right, ground, init).
object(box, left, none, init).

agent(monkey, left, ground, result(init,walk)).
object(box, left, none, result(init,walk)).

agent(monkey, left, box, result(result(init,walk),climb)).
object(box, left, monkey, result(result(init,walk),climb)).

$agent(agent\ name, agent\ position, stands\ on, situation)$
$object(object\ name, object\ position, who\ stands, situation)$

Action "effect" axiom for $walk(X, P_1, P_2)$:

$$\forall X, P_1, P_2, Z\ (agent(X, P_1, ground, Z) \wedge walks(X) \rightarrow agent(X, P_2, ground, result(Z, walk(X, P_1, P_2)))$$

Action "effect" axiom for $climb(X)$:

$$\forall X, P, Z\ (agent(X, P, ground, Z) \wedge object(box, P, none, Z) \rightarrow agent(X, P, box, result(Z, climb(X))) \wedge object(box, P, X, result(Z, climb(X)))$$

Frame problem

Action axioms describe how fulents change between situations
What happens to fluents, which are not used in the actions?
e.g., the objects while the agent walks

Frame problem: how to cope with the unchanged facts smartly

- many "frame" axioms may be necessary to express them in FOL

$$\forall X, V, W, Z, P_1, P_2$$
$$(object(X, V, Y, Z) \rightarrow object(X, V, Y, result(Z, walk(P_1, P_2))))$$

- f fluent predicates, a actions requires $O(f \cdot a)$ frame axioms
- many applications of axioms each step is computationally expensive
- some tricks diminish the problem, but it never goes away

Logical planning

We already have representations of **states** and **actions**
Goal of planning: logical representation of the desired state

$$\mathcal{G} \equiv \quad \exists Z \;\; agent(monkey, middle, box, Z)$$

Reasoning checks whether the goal formula follows from KB

$$KB \models \mathcal{G}$$

- knowledge base (KB) are the inference rules and the initial state
- reasoning finds a suitable Z or proves it does not exist
- desirable properties: **soundness**, **completeness**, **efficiency**
- reasoning procedures: **resolution**, deductive inference, etc.
  - see B0B01LGR
  - generally extremely computationally hard, possibly undecidable
  - the solution is correct, if reasoning successfully finishes
  - can be efficient and useful with **additional restrictions**

Domain Independent Automated Planning

Subfield of AI dealing (mainly) with

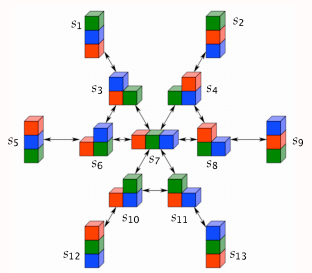
- representation languages with reasonable tradeoffs of expressivity and efficiency
- algorithms for finding plans for problems expressed in these languages

(The following slides are heavily based on Carmel Domshlak's slides)

Planning problems

What is in common?

- All these problems deal with action selection or control
- Some notion of problem state
- (Often) specification of initial state and/or goal state
- Legal moves or actions that transform states into other state



Planning task

For now focus on:

- Plans (aka solutions) are sequences of moves that transform the initial state into the goal state
- Intuitively, not all solutions are equally desirable

What is our task?

1. Find out whether there is a solution
2. Find any solution
3. Find an optimal (or near-optimal) solution
4. Fixed amount of time, find best solution possible
5. Find solution that satisfy property $\aleph$ (what is $\aleph$? you choose!)

Three Key Ingredients of Planning

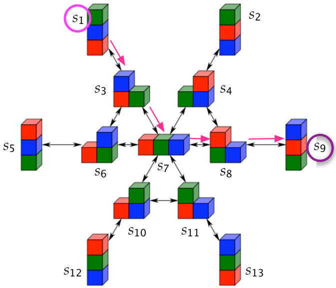
Planning is a form of general problem solving

`Problem` $\Longrightarrow$ `Language` $\Longrightarrow$ `Planner` $\Longrightarrow$ `Solution`

1. **models** for defining, classifying, and understanding problems
   - what is a *planning problem*
   - what is a *solution* (*plan*), and
   - what is an *optimal solution*
2. **languages** for representing problems
3. **algorithms** for solving them

Why planning is difficult?

- Solutions to planning problems are paths from an initial state to a goal state in the transition graph
- Dijkstra's algorithm solves this problem in $O(|V| \log(|V|) + |E|)$
- Can we go home??



- Solutions to planning

What is "classical" planning?

- dynamics: deterministic, nondeterministic or probabilistic
- observability: full, partial or none
- horizon: finite or infinite
- ...

1. classical planning
2. conditional planning with full observability
3. conditional planning with partial observability
4. conformant planning
5. Markov decision processes (MDP)
6. partially observable MDPs (POMDP)

Succinct representation of transition systems

- More compact representation of actions than as relations is often
  - possible because of symmetries and other regularities,
  - unavoidable because the relations are too big.
- Represent different aspects of the world in terms of different state variables. $\rightsquigarrow$ A state is a valuation of state variables.
- Represent actions in terms of changes to the state variables.

Planning Languages

Key issue

Models represented **implicitly** in a **declarative language**

Play two roles

- **specification**: concise model description
- **computation**: reveal useful info about problem's *structure*

The STRIPS language

A problem in STRIPS is a tuple $\langle P, A, I, G \rangle$

- P stands for a finite set of **atoms** (boolean vars)
- $I \subseteq P$ stands for **initial situation**
- $G \subseteq P$ stands for **goal situation**
- A is a finite set of **actions** a specified via pre(a), add(a), and del(a), all subsets of P

- States are collections of atoms
- An action a is applicable in a state s iff pre$(a) \subseteq s$
- Applying an applicable action a at s results in $s' = (s \setminus \text{del}(a)) \cup \text{add}(a)$

Why STRIPS is interesting?

- STRIPS operators are **particularly simple**, yet expressive enough to capture general planning problems.
- In particular, STRIPS planning is **no easier** than general planning problems.
- Many algorithms in the planning literature are **easier to present in terms of STRIPS**.

(The following example is based on Antonin Komanda's slides)

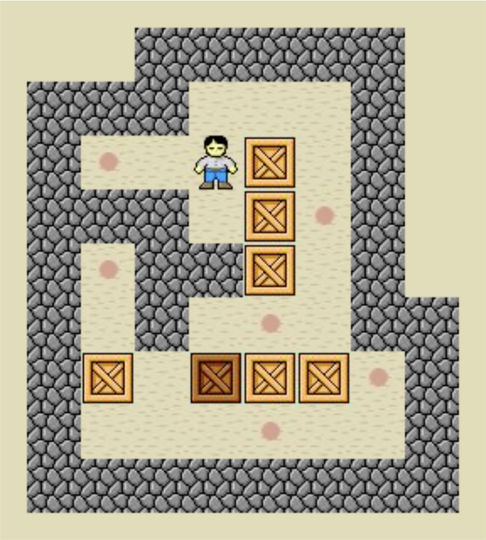
Sokoban - Example planning domain

State representation:

```
positions: a1, ... a6,...
           f1, ..., f2
box_at(P), free(P)
player_at(P)
adjacent(P1,P2)
adjacent2(P1,P2)
```

Operators (Actions):

```
move(X,Y):
  pre: player_at(X)
       adjacent(X,Y)
       free(Y)
  add: player_at(Y)
  del: player_at(X)

push(X, Y, Z):
  pre:  player_at(X)
        box_at(Y)
        free(Z)
        adjacent(X,Y)
        adjacent(Y,Z)
        adjacent2(X,Z)
  ...
```



Grounding of Actions

Operators (Actions):

```
move(X,Y):
  pre: player_at(X)
       adjacent(X,Y)
       free(Y)
  add: player_at(Y)
  del: player_at(X)

push(X, Y, Z):
  pre:  player_at(X)
        box_at(Y)
        free(Z)
        adjacent(X,Y)
        adjacent(Y,Z)
        adjacent2(X,Z)
  add: player_at(Y)
        box_at(Z)
        free(Y)
  del: player_at(X)
        box_at(Y)
        free(Z)
```

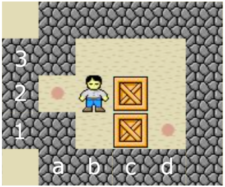
Grounding:

```
move_a1_a2
  pre: player_at_a1, adjacent_a1_a2, free_a2
  add: player_at_a2
  del: player_at_a1

move_a2_a3
  pre: player_at_a2, adjacent_a2_a3, free_a3
  add: player_at_a3
  del: player_at_a2

...

push_a1_a2_a3
  pre: player_at_a1, box_at_a2, free_a3
         adjacent_a1_a2, adjacent_a2_a3,
         adjacent_a1_a3
  add: player_at_a2, box_at_a3, free_a2
  del: player_at_a1, box_at_a2, free_a3
...
```

STRIPS Representation of Sokoban

A problem in STRIPS is a tuple $\langle P, A, I, G \rangle$

- P stands for a finite set of **atoms** (boolean vars)
- $I \subseteq P$ stands for **initial situation**
- $G \subseteq P$ stands for **goal situation**
- A is a finite set of **actions** a specified via pre(a), add(a), and del(a), all subsets of P




```
P = {player_at_a2, ..., player_at_d3,
     box_at_a2, ..., box_at_d3,
     free_a2, ..., free_d3,
     adjacent_a2_b2, ..., adjacent_d2_d3,
     adjacent2_a2_c2, ..., adjacent2_d1_d3 }

I = {player_at_b2, box_at_c1, box_at_c2,
     free_a2, free_b1, ..., free_d3,
     adjacent_a2_b2,..., adjacent_d2_d3, adjacent2_a2_c2,..., adjacent2_d1_d3}

G = {box_at_a2, box_at_d1}
```

Planning in Strips

We can just use A*:

- State: a set of true atoms
- Applicable actions: based on preconditions
- Action application: add the “add” atoms and delete the “del” atoms

  (No need for separate simulator implementation)

Problem structure allows **automated** construction of **heuristics**!

- Allows exploring general heuristics domain independently
- Simple heuristic: $h(s) = |G \setminus s|$
- Solve a suitable **simpler** version of the problem
- Abstraction: solve a smaller problem

  e.g., completely remove a predicate from the problem
- **Relaxation**: solve a less constraint problem
- Landmarks

Relaxation heuristics

Whole sub-field of planning in STRIPs and beyond

- Relaxation is a general technique for heuristic design:
  - Straight-line heuristic (route planning): Ignore the fact that one must stay on roads.
  - Manhattan heuristic (15-puzzle): Ignore the fact that one cannot move through occupied tiles.
- We want to apply the idea of relaxations to planning.
- Informally, we want to ignore bad side effects of applying actions.

Example (8-puzzle)

If we move a tile from x to y, then the good effect is (in particular) that x is now free.
The bad effect is that y is not free anymore, preventing us for moving tiles through it.

Relaxed planning tasks in STRIPS

In STRIPS, good and bad effects are easy to distinguish:

- Effects that make atoms true are good (add effects).
- Effects that make atoms false are bad (delete effects).

Idea for the heuristic: Ignore all delete effects.

Relaxed planning tasks in STRIPS

Definition (relaxation of actions)

The relaxation a^+ of a STRIPS action $a = \langle \mathsf{pre}(a), \mathsf{add}(a), \mathsf{del}(a) \rangle$ is the action $a^+ = \langle \mathsf{pre}(a), \mathsf{add}(a), \emptyset \rangle$.

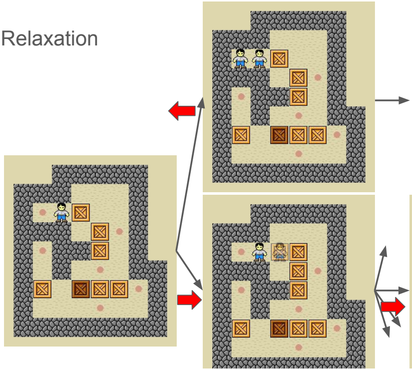
Definition (relaxation of planning tasks)

The relaxation Π^+ of a STRIPS planning task $\Pi = \langle P, A, I, G \rangle$ is the planning task $\Pi^+ := \langle P, \{a^+ \mid a \in A\}, I, G \rangle$.

Definition (relaxation of action sequences)

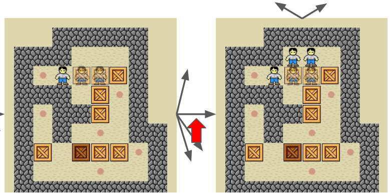
The relaxation of an action sequence $\pi = a_1 \dots a_n$ is the action sequence $\pi^+ := {a_1}^+ \dots {a_n}^+$.

Relaxation



```
move(X,Y):
  pre:  player_at(X)
        adjacent(X,Y)
        free(Y)
  add: player_at(Y)
  del: player_at(X)
```

```
push(X, Y, Z):
  pre:  player_at(X)
        box_at(Y)
        free(Z)
        adjacent(X,Y)
        adjacent(Y,Z)
        adjacent2(X,Z)
  add: player_at(Y)
        box_at(Z)
        free(Y)
  del: player_at(X)
        box_at(Y)
        free(Z)
```



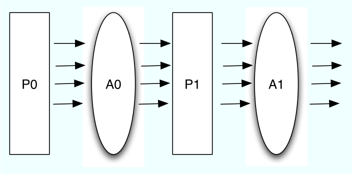
Building Relaxed Planning Graph

Computing the optimal relaxed plan is still NP hard

But we can do something simpler

- Build a layered reachability graph $P_0, A_0, P_1, A_1, \ldots$

$$\begin{aligned} P_0 &= \{p \in I\} \\ A_i &= \{a \in A \mid \mathsf{pre}(a) \subseteq P_i\} \\ P_{i+1} &= P_i \cup \{p \in \mathsf{add}(a) \mid a \in A_i\} \end{aligned}$$



- Terminate when $G \subseteq P_i$

Example

$$I = \{a = 1, b = 0, c = 0, d = 0, e = 0, f = 0, g = 0, h = 0\}$$

$$a_1 = \langle \{a\}, \{b, c\}, \emptyset \rangle$$

$$a_2 = \langle \{a, c\}, \{d\}, \emptyset \rangle$$

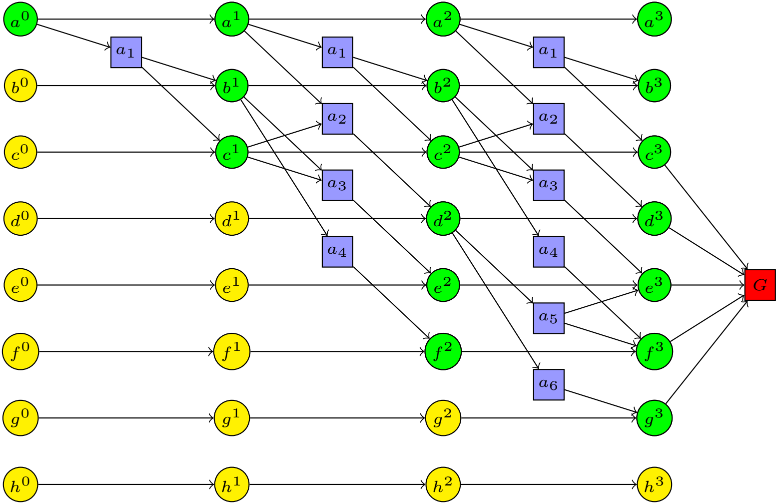
$$a_3 = \langle \{b, c\}, \{e\}, \emptyset \rangle$$

$$a_4 = \langle \{b\}, \{f\}, \emptyset \rangle$$

$$a_5 = \langle \{d\}, \{g\}, \emptyset \rangle$$

$$G = \{c = 1, d = 1, e = 1, f = 1, g = 1\}$$

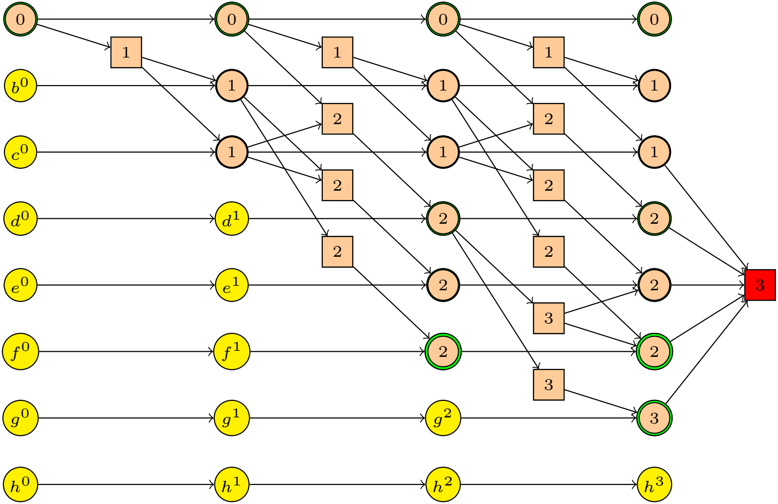
Relaxed Planning Graph



Forward cost heuristic h_{max}

- Propagate cost layer by layer from start to goal
- At actions, take maximum cost of achieving preconditions $+1$
- At propositions, take the cheapest action to achieve it

Computing heuristic h_{max}



Summary

Logic is a powerful language for describing AI problems

Situation calculus is a logical formalism for reasoning about situations developing in time

Of-the-shelf logical reasoning methods are usable for planning

However, expressivity goes against efficiency

AI planning creates logical representations and algorithms specially designed for planning

STRIPS is a simple, but powerful language for representing planning problems

Logical representation of problems allows automated construction of A* heuristics