A Concise Introduction to Automated Planning and PDDL

Andrea Marrella

marrella@diag.uniroma1.it

Reasoning Robots: Reasoning about Action in Cognitive Robotics

March 20, 2018

SAPIENZA
UNIVERSITÀ DI ROMA

AI and Autonomous Behaviour

- The challenge of building devices that act autonomously <u>is at the center</u> of the AI research from its origins.
- At the center of the problem of autonomous behavior is the **control problem** (or **action selection problem**).
  - *specify a **controller** that selects the action to do next*
- Traditional hard-coded solutions specify a **pre-scripted controller** in a high-level language.
  - ✓ They do not suffer combinatorial explosion.
  - The burden is all put on the programmer.
  - Hard-coded solutions are usually biased and <u>*tend to constraint the search*</u> in some way.
- The question of action selection for AI researchers is:
  - *What is the best way to intelligently constrain this search?*

AI and Autonomous Behaviour

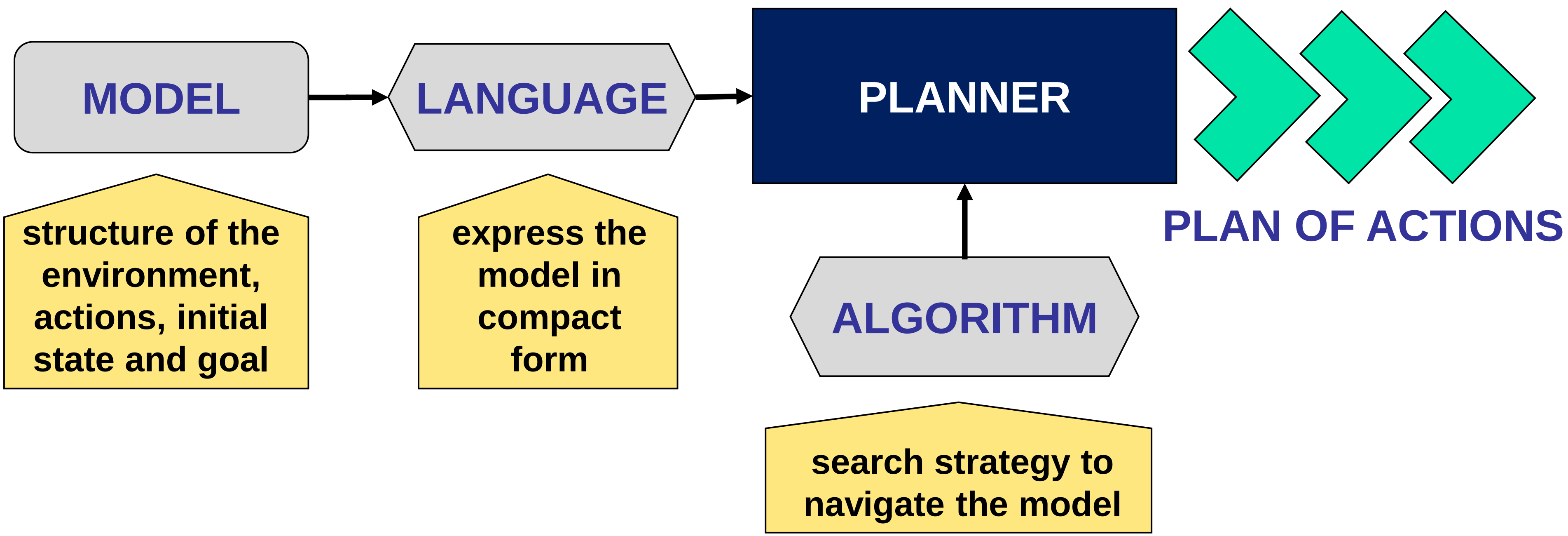
- Two approaches in AI to tackle autonomous behavior:
- **Learning-based approach**
  - The controller is **learnt from experience**.
    - ✓ Discovery and interpretation of meaningful patterns for a given task.
    - – Learned solutions are usually black-box.
- **Model-based approach**
  - The controller is **derived automatically** from a model of the domain of interest, the actions, the current state, and the goal.
    - ✓ The models are all conceived to be general.
    - – The problem of solving a model is computationally intractable.

In this lecture, we introduce the basic ingredients of automated planning and the PDDL language for representing planning problems.

Automated Planning

- In AI, **automated planning** is conceived as the:

model-based approach for the automated synthesis of plans of actions to achieve goals.



Planning Models

- **Several classes of planning models**, which depend on the properties of the problems to be represented:
  - full or partial observability of the current state;
  - uncertainty in the initial state (fully or partially known);
  - uncertainty in the actions dynamics (deterministic or not);
  - uncertainty represented by sets of states or probability distributions;
  - the type of feedback (full, partial or no state feedback).

Planning is computationally <u>intractable</u> even for the simplest models…

..BUT..

…classical planners <u>solve efficiently</u> real problems with hundreds of propositions!

Classical Planning Model

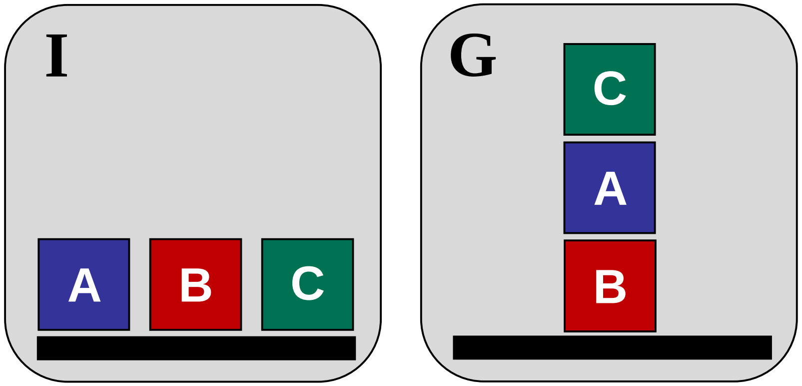
- **finite** and **discrete** state space S
- a **known initial state** $I \in S$
- a set $S_G \subseteq S$ of **goal states**
- **actions** $A(s) \subseteq A$ applicable in each $s \in S$
- a **deterministic transition function** $s' = f(a, s)$ for $a \in A(s)$
- positive **action costs** $c(a,s)$

- A **solution** or **plan** is a sequence of applicable actions $\pi = a_0, \ldots, a_n$ that maps I into S_G
  - There are states $s_0, \ldots, s_{n+1}$ such that $s_{i+1} = f(a_i, s_i)$ and $a_i \in A(s_i)$ for $i = 0,\ldots, n$ and $s_{n+1} \in S_G$

- A plan is **optimal** if it minimizes the sum of action costs $\sum_{i=0,\ldots,n} c(a_i, s_i)$. If costs are all 1, plan cost is plan length.

Example: The Blocks World Domain

- Given a set of blocks of various colors sitting on a table, the goal is to build one or more vertical stacks of these blocks.
- Initial state: **I**
- Goal: **G**



- Available actions: moving a block
  - from the table to the top of another block
  - from the top of another block to the table
  - from the top of one block to the top of another block

Planning Domain Definition Language

- The standard representation language for automated planners is known as the **Planning Domain Definition Language** (PDDL).

- Components of a PDDL planning task:
  - **Objects**: Things in the world that interest us.
  - **Predicates**: Properties of objects that we are interested in; they can be true or false.
  - **Initial state**: The state of the world that we start in.
  - **Goal specification**: Things that we want to be true.
  - **Actions/Operators**: Ways of changing the state of the world.

Planning Domain Definition Language

- Problems in PDDL are expressed in two separate parts:
  - PDDL **Planning Domain PD** (available actions and predicates representing explicit representation of the world).
  - PDDL **Planning Problem PR** (objects, initial state **I** and goal condition **G**).

  - A planner that takes in input a problem encoded in PDDL is said to be **domain-independent**, since it is able to automatically produce a plan without knowing what the actions and domain stand for.

  - PDDL provides the ground for performing a **direct comparison** between different planning techniques and algorithms and evaluating against classes of problems.

Domain files

- Domain files look like this:

```
(define (domain <domain name>)
  <PDDL code for predicates>
  <PDDL code for first action>
  [...]
  <PDDL code for last action>
)
```

  - `<domain name>` is a string that identifies the planning domain, e.g., `blocks-world`.
  - Example on the web: `blocks-world.pddl`

Problem files

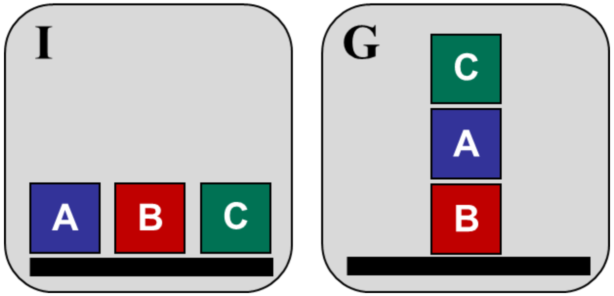
- Problem files look like this:

```
(define (problem <problem name>)
  (:domain <domain name>)
   <PDDL code for objects>
   <PDDL code for initial state>
   <PDDL code for goal specification>
)
```

  - `<problem name>` is a string that identifies the planning task, e.g. `blocks-world-3`.
  - `<domain name>` must match the domain name in the corresponding domain file.
  - Example on the web: `blocks-world-3.pddl`

Example: The Blocks World Domain

- **Objects**: The blocks locations. Blocks can be on the table or on top of another block. Four blocks for the specific instance.



- **Predicates**: Is a block clear (i.e., with no block on top)? Does a block have another block on top of it?
- **Actions/Operators**: Clear blocks can be moved on top of another block or on the table, respectively.
- **Initial state**: Blocks A, B and C are initial arranged on the table.
- **Goal specification**: re-arrange the blocks so that C is on A and A is on B.

The Blocks World in PDDL

Planning Domain

```
(define (domain blocks-world)
(:requirements :strips)
(:objects block)
(:predicates (on ?x ?y - block)
             (clear ?x - block))
```

Objects of the domain and **predicates** describe the **state** of the world.

The Blocks World in PDDL

Planning Domain

```
(:action move
 :parameters (?b ?x ?y - block)
 :precondition (and (on ?b ?x)
                    (clear ?b) (clear ?y))
 :effect (and (not (on ?b ?x)) (not (clear ?y))
              (on ?b ?y) (clear ?x)))
```

Actions are described in terms of **preconditions** under which an action can be executed, and **effects** on the state of the world**.**

Both preconditions and effects are stated in terms of the predicates.

The Blocks World in PDDL
Planning Domain

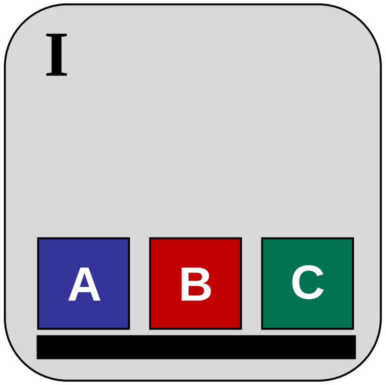
```
(:action moveToTable
 :parameters (?b ?x - block)
 :precondition (and (on ?b ?x) (clear ?b))
 :effect (and (on ?b table) (clear ?x)
              (not (on ?b ?x)))
)
```

Action `moveToTable` is necessary to properly represent the fact that the table does not have to be clear to move a block onto it.

The Blocks World in PDDL
Planning Problem

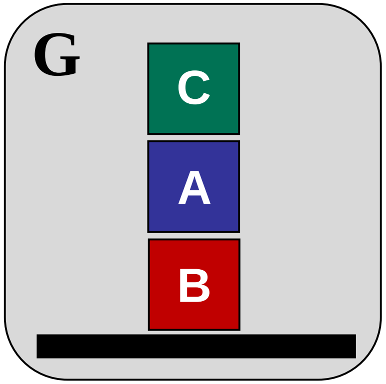
Objects

```
(:objects A B C table - block)
```



Initial State I

```
(:init
 (on A table) (clear A)
 (on B table) (clear B)
 (on C table) (clear C))
```

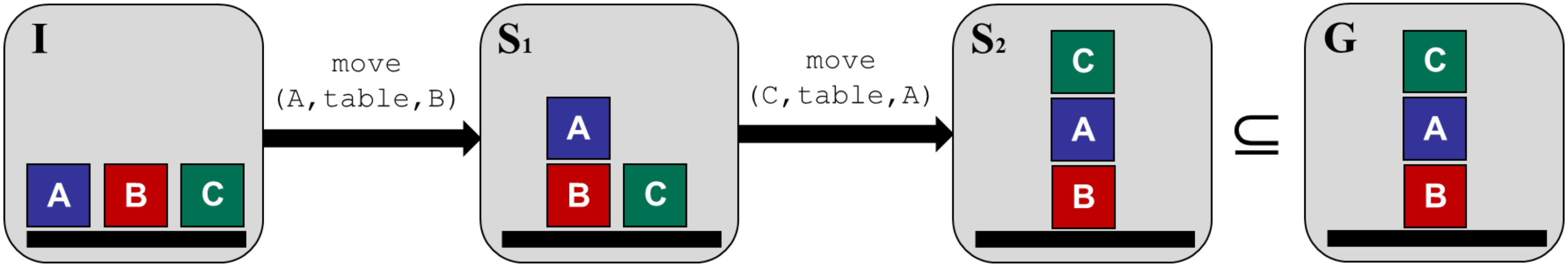


Goal G

```
(:goal
 (and (on C A) (on A B))
)
```

The Blocks World in PDDL

Optimal Plan

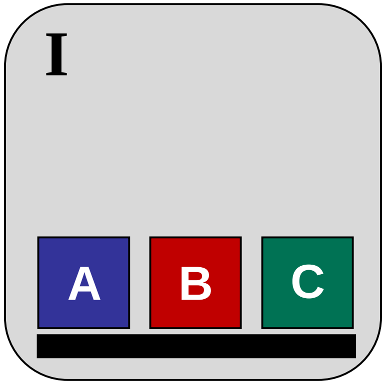


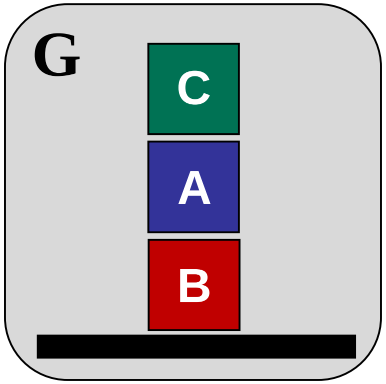

```
Begin plan
1.  (move A table B)
2.  (move C table A)
End plan
```

Since S2 is a state satisfying the goal G, the solution found is a **valid plan**.

The **quality of a solution** depends by the specific search algorithm employed by the planner.

```
Begin plan
1.  (move B table A)
2.  (move B A table)
3.  (move A table B)
2.  (move C table A)
End plan
```

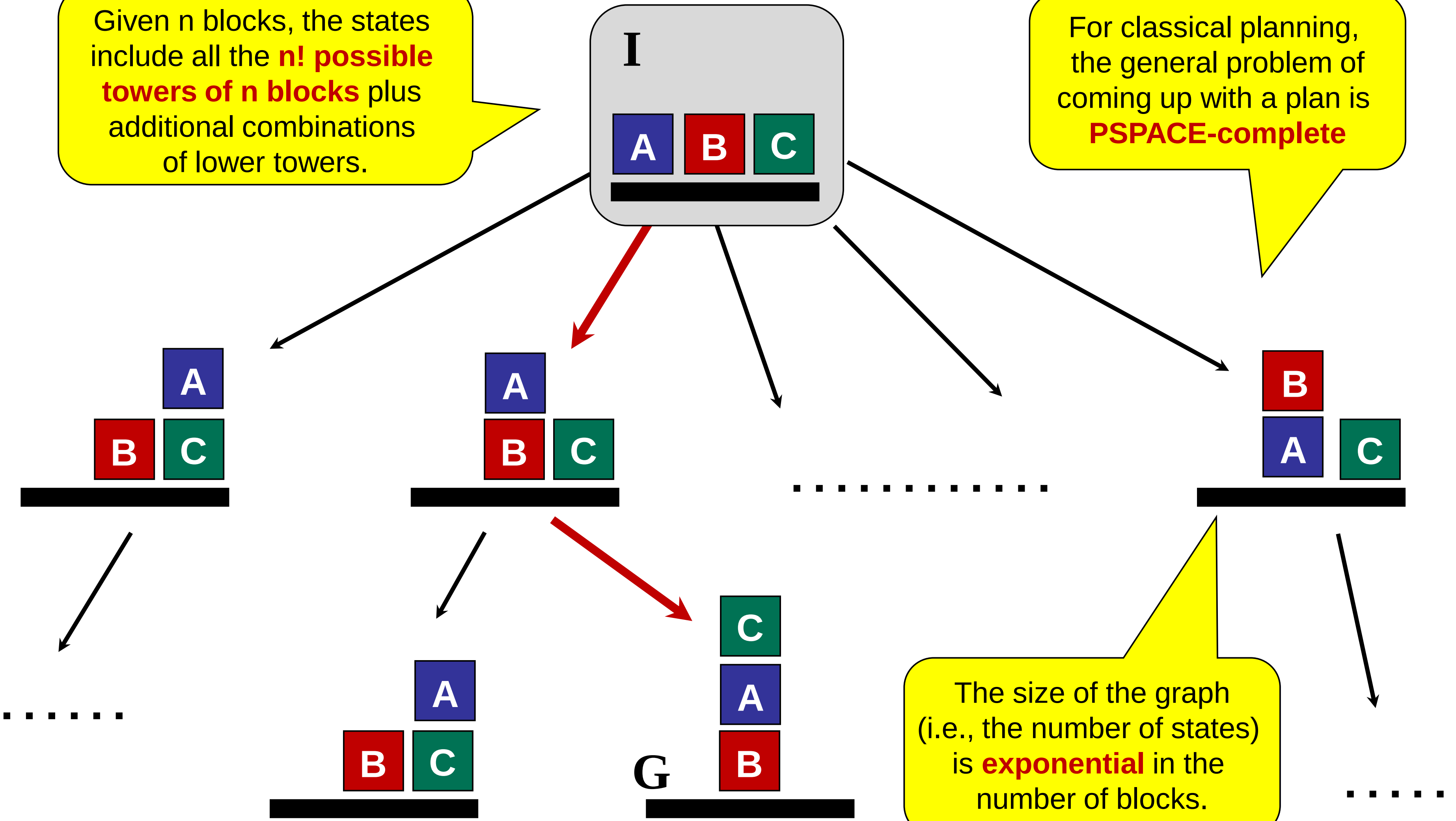




Graph of a planning problem

Given n blocks, the states include all the **n! possible towers of n blocks** plus additional combinations of lower towers.

For classical planning, the general problem of coming up with a plan is **PSPACE-complete**



The size of the graph (i.e., the number of states) is **exponential** in the number of blocks.

Challenge of automated planning

- **Challenge**: achieving **both** **generality** and **scalability**.
  - **Generality**: A planner can solve ***arbitrary problem instances.***
    - A planner does not know what the actions, and domain stand for.
    - This is very different from writing a *domain-specific* solver.
  - **Scalability**: Planners embed very effective **domain-independent heuristics** to drive the searching task towards the goal.
    - An ***heuristic function*** provides an estimate of the cost to reach the goal from the current state (Examples: Best-First Search, A*, Hill Climbing, etc).
- State-of-the art planners** provide **customized implementations** of the search algorithms with different properties of completeness, optimality, and memory complexity.

**Cf. http://icaps-conference.org/index.php/main/competitions

Extensions of PDDL

- Several extensions of PDLL:
  - **PDDL 1.2**: Base version of the language. Among the basic constructs, it includes **STRIPS**, **ADL** and **conditional effects**.
  - **PDDL 2.1**: It introduces **numeric fluents** (e.g., to model non-binary resources such as time, distance, weight, etc.), **plan-metrics** (to allow *quantitative evaluation* of plans, and not just goal-driven), and **durative/continuous actions** (which could have variable, non-discrete length, conditions and effects).
  - **PDDL 2.2**: It introduces **derived predicates** (to model the dependency of given facts from other facts), and **timed initial literals** (to model exogenous events occurring independently from plan-execution).
  - **PDDL 3.0:** It introduces **preferences** (hard- and soft-constraints, in form of logical expressions, to be satisfied in specific points of the plan).
  - **PDDL 3.1:** It introduces **object fluents** (functions' range can be any object-type).

Notes on action effects

- In the base version of PDDL (v1.2), action effects can be more complicated than seen so far.
- They can be **universally quantified**:

```
(forall (?v1 ... ?vn))
    <effects>
```

- They can be **conditional**:

```
(when <condition>
          <effect>)
```

- We now investigate a **concrete problem** (*trace alignment in process mining*) solved by planning techniques that require the use of conditional effects and universal quantification in actions effect.

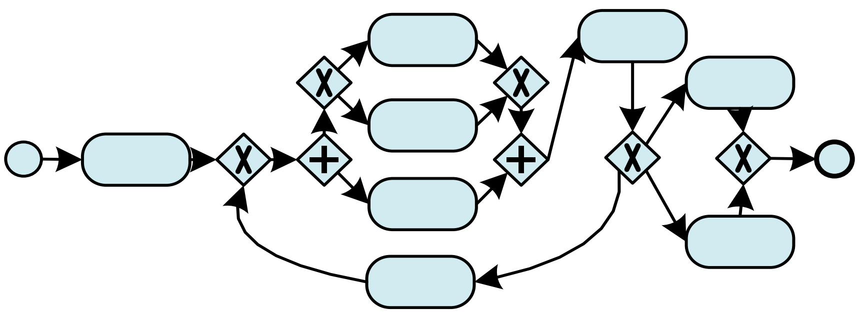
The Blocks World in PDDL
Conditional effects

```
(:action move
 :parameters (?b ?x ?y - block)
 :precondition (and (on ?b ?x) (clear ?b))
 :effect (and (not (on ?b ?x)) (clear ?x))
          (when (clear ?y)
                (and (on ?b ?y) (not (clear ?y))))
          (when (not (clear ?y))
                (on ?b table))
          )
)
```

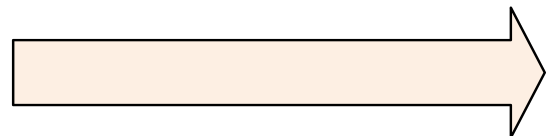
Using conditional effects allows to **model a single planning action** that represent blocks movement.

Conformance Checking

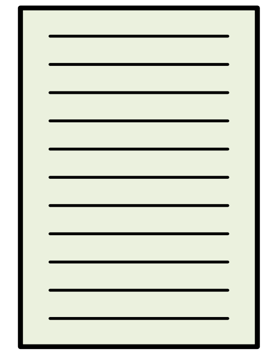
Any execution of a process model produces a new **execution trace** (i.e., a **process instance**) recorded in an **event log.**



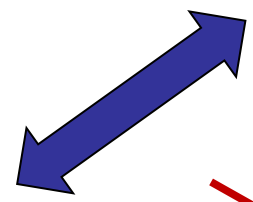
process model



Execution by a Process Management System



event log



regulations

Concrete process executions are compliant with regulations and laws?

The trace alignment problem

- Process models are typically not enforced by information systems (human behavior is often involved).
  - Traces can be **dirty**, with ***spurious*** or ***missing events***.
- **Trace alignment** is the problem of **cleaning** such dirty traces against process models to the aim of:
  - verify if a trace is **compliant** with its underlying process model;
  - identifying the **root** and the **severity** of each deviation;
  - **repairing the trace** to make it compliant with the process model.
- The existing techniques to compute optimal alignments
  - provide **ad-hoc implementations** of the A* algorithm.
  - **do not scale efficiently** when process models and event logs are of considerable size.

SOLUTION: The problem of computing optimal alignments can be formulated as a **planning problem** in PDDL, which employs **conditional effects** and **universal quantifiers**.

Trace alignment

- Given a trace t and a DECLARE model D (which is used to define the regultions) find the optimal alignment of t with respect to D.
  - A DECLARE model $D = (A, \pi_D)$ consists of a set of activities A involved in a process and a collection of ***temporal constraints*** π_D defined over A.
  - DECLARE constraints (aka ***templates***) define parameterized classes of properties and enjoy a precise semantics in **LTL$_f$** (LTL over finite traces).

Existence(A)
LTL Formalization: $\Diamond$**A**
A occurs at least 1 time.
BCAAC ✓ *BCC* ✗

Absence(C)
LTL Formalization: **¬$\Diamond$C**
A never occur.
BAA ✓ *BCAC* ✗

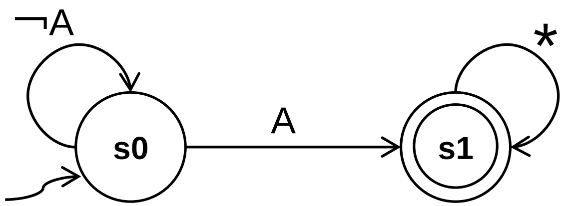
Response(A, B)
LTL Formalization: $\Box$ **(A→$\Diamond$B)**
If *A* occurs, then *B* occurs after *A*.
BCAAC ✗ *CAACB* ✓ *BCC* ✓

From LTL$_f$ to DFAs

- For any **LTL$_f$** formula there exists a DFA that accepts all the traces satisfying the formula.

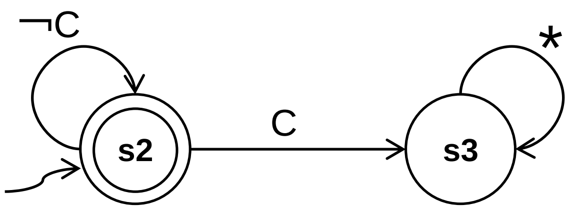
<u>Existence(A)</u>

LTL Formalization: ◊**A**
A occurs at least 1 time.
BCAAC ✓ *BCC* ✗



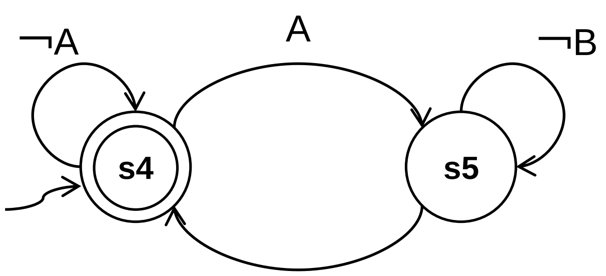
<u>Absence(C)</u>

LTL Formalization: **¬◊C**
A never occur.
BAA ✓ *BCAC* ✗



<u>Response(A, B)</u>

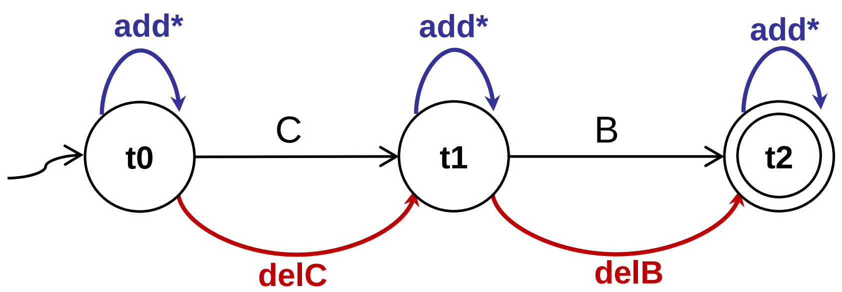
LTL Formalization: □ **(A→◊B)**
If *A* occurs, then *B* occurs after *A.*
BCAAC ✗ *CAACB* ✓ *BCC* ✓



Automata-based solution

G. De Giacomo, F. M. Maggi, A. Marrella, F. Patrizi, *On the Disruptive Effectiveness of Automated Planning for LTLf-based Trace Alignment*. AAAI 2017

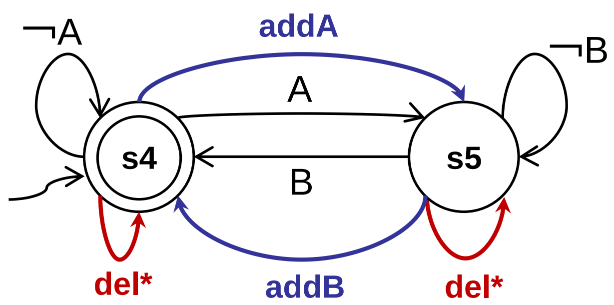
- Trace alignment can be solved using automata:
  - One automaton for the trace (***trace automaton***).



  - Accepts input trace (<C,B>) plus all other traces, however…
  - …changes wrt. input trace must be marked by add/del, e.g.,
    - <C,B,C> = C B addC
    - <B,C,B,B> = delC B addC addB addB
  - Adds and dels have (possibly different) positive costs.

Automata-based solution

- One automaton per constraint (***constraint automaton***) **augmented** to account for adds and dels.

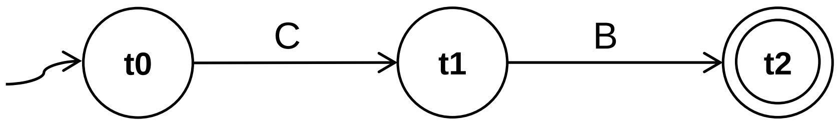


- Accepts all (possibly repaired) traces satisfying the constraint.

- An **alignment** is a sequence of **syncronous steps** performed in all *augmented constraint automata* and in the *augmented trace automaton* such that -- at the end of the alignment -- each automaton is **in at least one** accepting state.

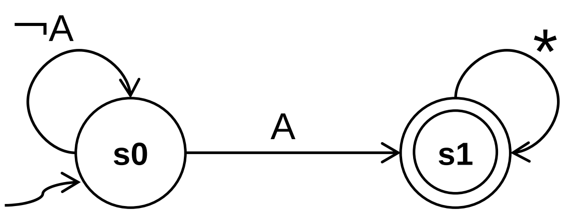
An example of Trace Alignment

Trace: <C,B>

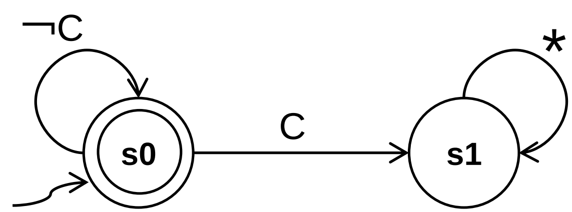


LTL$_f$ Constraints

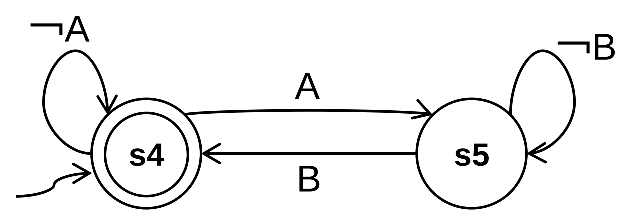
<u>Existence(A):</u> ◊A



<u>Absence(C):</u> ¬◊C

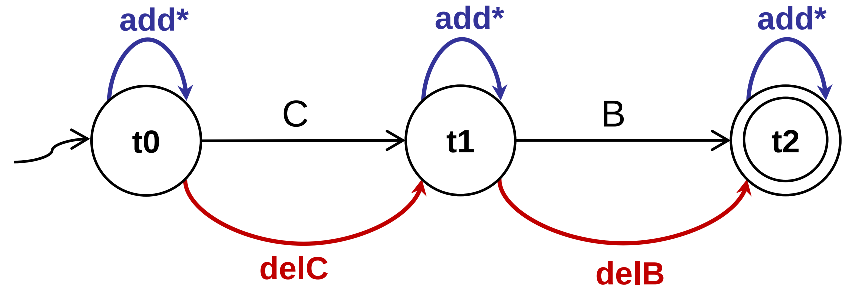


<u>Response(A,B):</u> □(A→◊B)



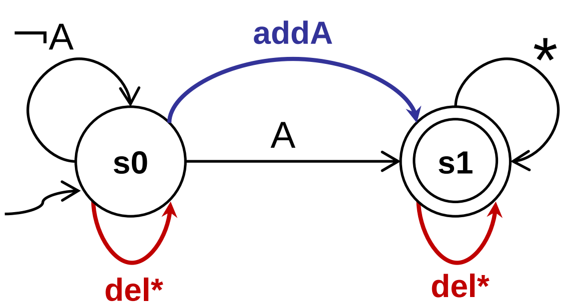
An example of Trace Alignment

Trace: <C,B>



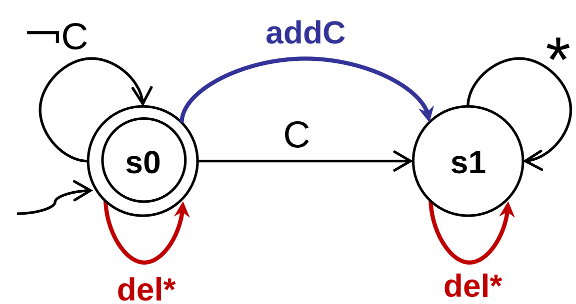
LTL$_f$ Constraints

Existence(A): ◊A

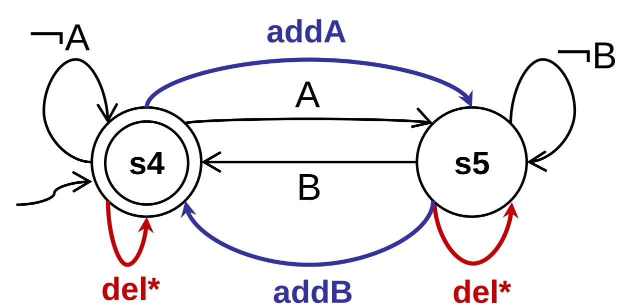


Augmented trace automaton and **augmented constraint automata**

Absence(C): ¬◊C



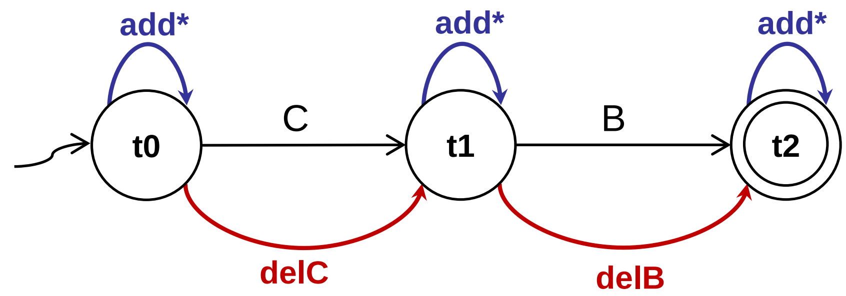
Response(A,B): □(A→◊B)



An example of Trace Alignment

Trace: <C,B>

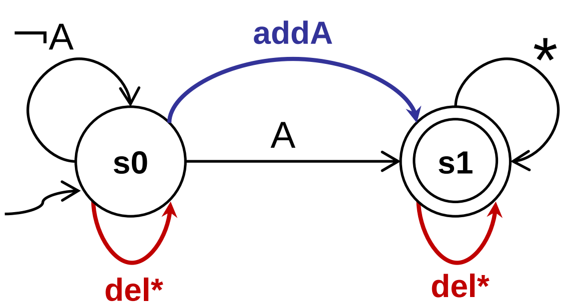
Optimal Plan: <delC,addA,B>



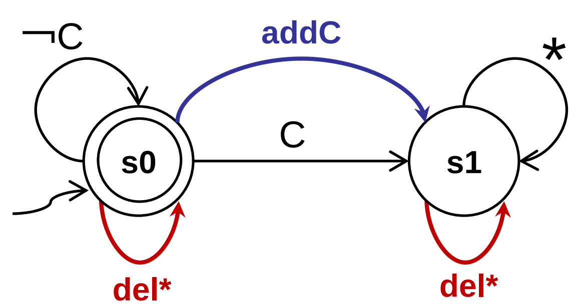
LTL$_f$ Constraints

…if **adds** and **dels** have **unitary cost**.

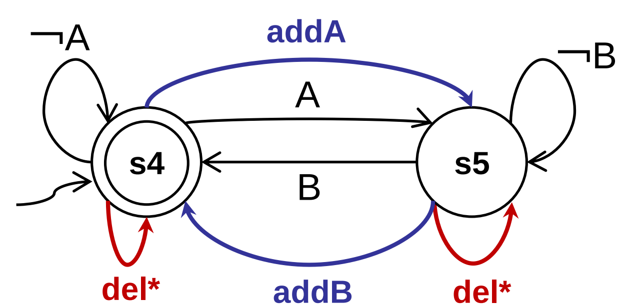
Existence(A): ◊A



Absence(C): ¬◊C



Response(A,B): □(A→◊B)



Trace alignment problem in PDDL

- The automata-based approach can be recast as a **cost-optimal planning problem** using PDDL.
  - **Planning Domain:**
    - Input events modeled by **synchronization actions** with null cost.
    - **Adds** and **dels** modeled by planning actions with positive costs.
    - **Domain propositions** encode the structure and the dynamics of the augmented trace and of all augmented constraint automata.
  - **Problem:**
    - **Initial state**: all automata in their starting state.
    - **Goal state**: all automata in (at least one) final state.
  - **Solution:**
    - **Optimal** (i.e., **minimal-cost**) **plan** to reach the goal state.

PDDL Planning Domain

Boolean Predicates

```
(types trace_state automaton_state - state activity)
```

It captures the activities involved in a transition between two states of a constraint/trace automaton.

They identify the states of any constraint automaton and of the trace automaton.

```
(:predicates
 (trace ?t1 - trace_state
        ?e - activity
        ?t2 - trace_state)
 (automaton ?s1 - automaton_state
            ?e - activity
            ?s2 - automaton_state)
 (cur state ?s - state)
 (final state ?s - state)
)
```

They hold if there exists a transition in the trace/constraint automaton from two states, being `e` the activity involved in the transition.

They hold if `s` is the current/accepting state of a trace/constraint automaton.

PDDL Planning Domain

Sync action

It is applied only if there exists a transition from the current state `t1` of the trace automaton to a subsequent state `t2`, being `e` the activity involved in the transition. The action **has no cost**, as it stands for no change in the trace.

```
(:action sync
 :parameters (?t1 - trace_state ?e - activity
              ?t2 - trace_state)
 :precondition (and (cur_state ?t1) (trace ?t1 ?e ?t2))
 :effect(and (not (cur_state ?t1)) (cur_state ?t2)
             (forall (?s1 ?s2 - automaton_state)
             (when (and (cur_state ?s1)
                        (automaton ?s1 ?e ?s2))
             (and (not (cur_state ?s1))
                       (cur_state ?s2))))))
```

CONDITIONAL EFFECT: The action is performed in each constraint automaton for which there exists a transition involving the activity `e` that connects `s1` – the current state of the automaton – with a different state `s2`.

PDDL Planning Domain

Add action

Add actions make total cost of the alignment increasing of a predefined value.

```
(:action add
 :parameters (?e - activity)
 :effect (and (increase (total-cost) 1)
              (forall (?s1 ?s2 - automaton_state)
                      (when (and (cur_state ?s1)
                                 (automaton ?s1 ?e ?s2))
                            (and (not (cur_state ?s1))
                                 (cur_state ?s2))))))
```

CONDITIONAL EFFECT: The action is performed only for transitions involving the activity `e` between two different states of any constraint automaton, with the current state of the trace automaton that remains the same after the execution of the action.

PDDL Planning Domain

Del action

Del actions make total cost of the alignment increasing of a predefined value.

```
(:action del
 :parameters (?t1 - trace_state ?e - activity ?t2 - trace_state)
 :precondition (and (cur_state ?t1) (trace ?t1 ?e ?t2))
 :effect(and (increase (total-cost) 1)
              (not (cur_state ?t1)) (cur_state ?t2)))
```

It yields a single move in the trace automaton.

Initial and Goal State in PDDL

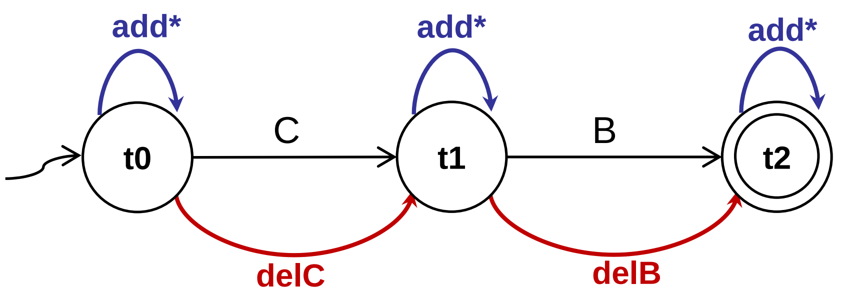
```
(:objects
 t0 t1 t2 - trace_state
 s4 s5 - automaton_state
 A B - activity)

(:init
(= (total-cost) 0)

(cur_state t0)
(trace t0 C t1)
(trace t1 B t2)
(final_state t2)

(cur_state s4)
(automaton s4 A s5)
(automaton s5 B s4)
(final_state s5))

(:goal (forall (?s - state)
               (imply (cur_state ?s)(final_state ?s))))

(:metric minimize (total-cost))
```

Representation of the trace automaton.

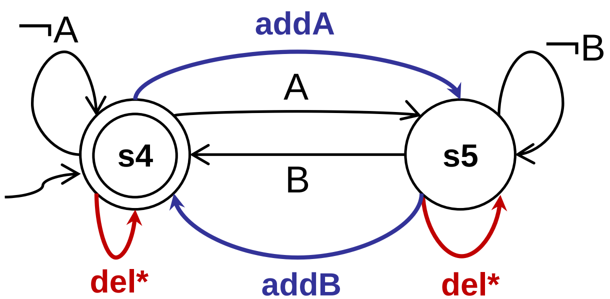
Representation of the constraint automaton.

Minimization of the total cost of the alignment.

Trace



□ (A→◊B):



Concluding Remarks

- Planning models are **all general** in the sense that they are **not bound** to specific problems or domains.
- This **generality** is coupled with the notion of **intelligence** which requires the ability to deal with new problems.
- The price for generality is **computational**:
  - planning over models represented in compact form is **intractable** in the worst case, yet currently large classical problems can be solved **very quickly**.
- **Suggested reading and resources**
  - **Fast Downward planning system**: http://www.fast-downward.org/
  - **Int. Plan. Comp.**: http://www.icaps-conference.org/index.php/Main/Competitions
  - **Book:** Hector Geffner, Blai Bonet: *A Concise Introduction to Models and Methods for Automated Planning*. Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers 2013, ISBN 9781608459698