Automated Planning for BPMers: Research Challenges and Successful Applications

Andrea Marrella

marrella@diag.uniroma1.it

Tutorial @ AI4BPM Bridge 2023

February 8, 2023

SAPIENZA
UNIVERSITÀ DI ROMA

Syllabus

1. Towards AI-Augmented BPM with Planning
2. Basics of Automated Planning
3. Automated Planning for BPM
4. Planning-based Declarative Trace Alignment
5. Conclusions

Syllabus

1. **Towards AI-Augmented BPM with Planning**
2. Basics of Automated Planning
3. Automated Planning for BPM
4. Planning-based Declarative Trace Alignment
5. Conclusions

Towards AI-Augmented BPM

- BPM research is expanding towards new challenging domains (healthcare, smart manufacturing, etc.) characterized by:
  - ever-changing requirements;
  - unpredictable and cyber-physical environments;
  - increasing amounts of data that influence the running processes.

- BPM systems need techniques that go **beyond hard-coded solutions** and are capable of **autonomous behavior**.

M. Dumas, F. Fournier, L. Limonad, A. Marrella, M. Montali, et al. **AI-Augmented Business Process Management Systems: A Research Manifesto**. *ACM Trans. on Management Information Systems, Volume 14, Issue 1 (2023)*

- The challenge of building physical devices that act autonomously is **at the center of the AI research** from its origins.

AI and Autonomous Behaviour

- 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 (usually) do not suffer combinatorial explosion.
  - − The burden is all put on the programmer.
  - − Hard-coded solutions are usually problem-dependent 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?*

Model-based approaches in AI

- **Model-based approaches** to tackle autonomous behavior:
  - 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.

Automated Planning

H. Geffner, B. Bonet, **A Concise Introduction to Models and Methods for Automated Planning**. *Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool (2013)*

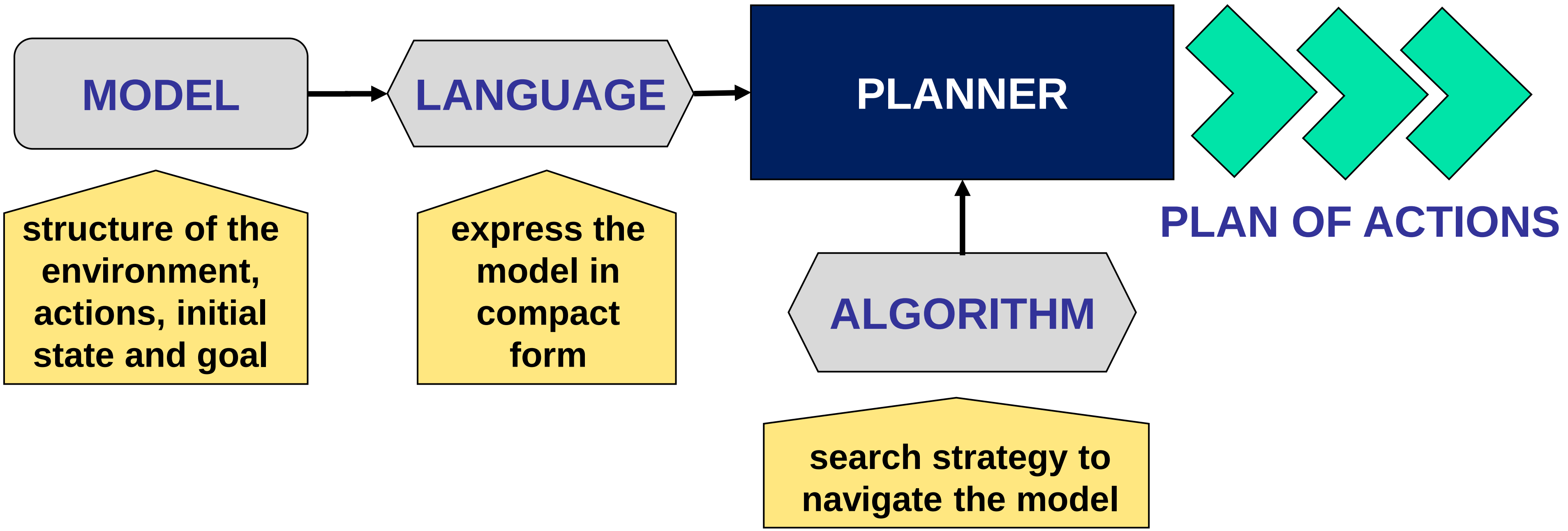
Syllabus

1. Towards AI-Augmented BPM with Planning
2. **Basics of Automated Planning**
3. Automated Planning for BPM
4. Planning-based Declarative Trace Alignment
5. Conclusions

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 π = a_0 , … , a_n that maps I into S_G
  - There are states s_0 , … , 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.

Planning Domain Definition Language

- The standard representation language for planners is 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.
  - **Functions:** Variables that apply to zero or more objects and are assigned with a numeric value.
  - **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 produces a plan without knowing what the actions and domain stand for.

Domain files

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

- `<domain name>` is a string that identifies the planning domain, e.g., `petri-net`.

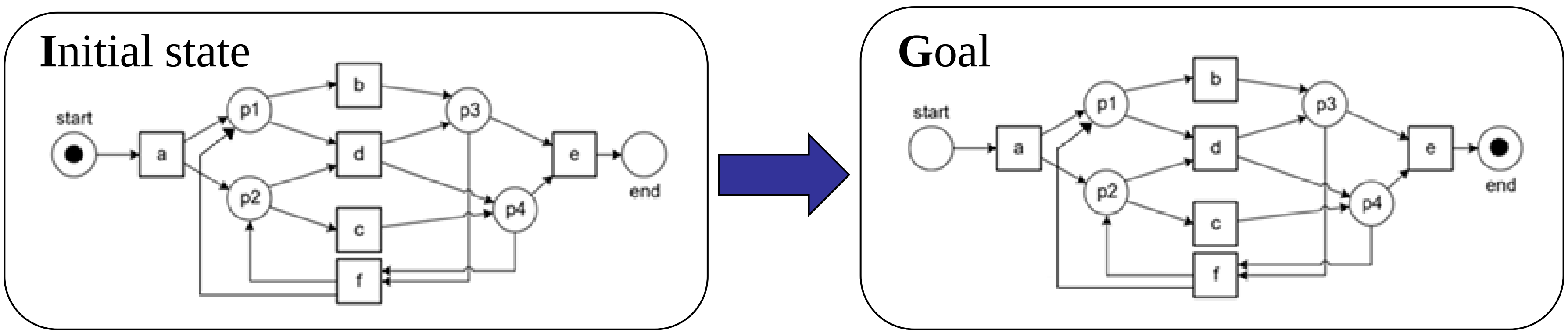
Problem files

```
(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. `reachability-problem`.
- `<domain name>` must match the domain name in the corresponding domain file, e.g., `petri-net`.

Example: Reachability in Petri Nets

- Given a Petri Net (PN) with an initial marking m_0 and a target marking m_n, if there exists a sequence of transition firings $<t_1 \ldots t_n>$ that leads from m_0 to m_n, then m_n is said to be **reachable** from m_0



- One available action: **firing**
  - If t is *enabled*, firing t changes the marking of the PN.
    - Each token from the input places of t is consumed, and one token is produced in any output place of *t*.

Example: The reachability problem

- **Objects**: Places and Transitions. For the specific case study, 6 places and 6 transitions.
- **Predicates**: Is a place an input place (or an output place) of a transition? Does a place contain a token?
- **Functions:** `total-cost` to keep track the cost of the plan under construction.
- **Actions/Operators**: Firing of a transition t.
- **Initial state**: A token is in place `start`. The other places do not contain any token.
- **Goal specification**: A token is in place `end`. The other places do not contain any token.

PDDL Editor

- **Planning.Domains**
  - URL: http://planning.domains/

The Reachability problem in PDDL
Planning Domain

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

```
(define (domain petri-net)
   (:types place transition)
   (:predicates (token ?p - place)
                (input_place ?t - transition ?p - place)
                (output_place ?t - transition ?p - place))
   (:action fire
       :parameters (?t - transition)
       :precondition (forall (?p - place)
                        (imply (input_place ?t ?p)
                               (token ?p)))
       :effect (and (forall (?p - place)
                      (when (input_place ?t ?p)
                            (not (token ?p))))
                    (forall (?p - place)
                      (when (output_place ?t ?p)
                            (token ?p)))
                    (increase (total-cost) 1)))
)
```

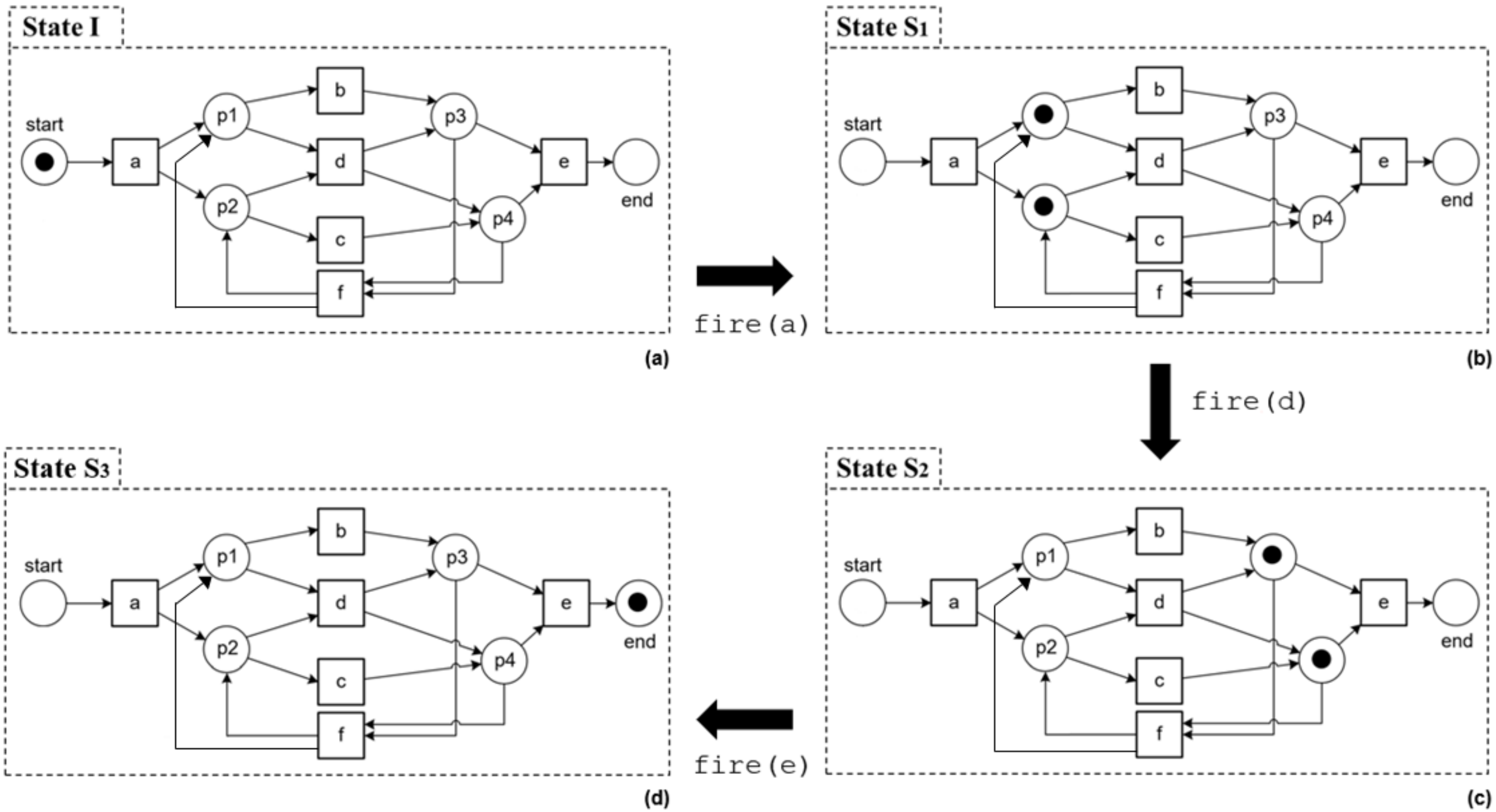
Actions are described in terms of **preconditions** under which an action can be executed, and **effects** on the state of the world, stated in terms of the predicates.

The Reachability problem in PDDL
Planning Problem

```
(define (problem pr1)
   (:domain petri-net)
   (:objects start p1 p2 p3 p4 end - place
                  a b c d e f - transition)
   (:init (token start) (input_place a start)
          (output_place a p1) (output_place a p2)
          (input_place b p1) (output_place b p3)
          (input_place c p2) (output_place c p4)
          (input_place d p1) (input_place d p2)
          (output_place d p3) (output_place d p4)
          (input_place e p3) (input_place e p4)
          (output_place e end) (input_place f p3)
          (input_place f p4) (output_place f p2)
          (= (total-cost) 0))
   (:goal (and (token end) (not (token start))
               (not (token p1)) (not (token p2))
               (not (token p3)) (not (token p4))))
   (:metric minimize (total-cost))
)
```

The Reachability problem in PDDL
Optimal plan

```
Begin plan
1. (fire a)
2. (fire d)
3. (fire e)
End plan
```



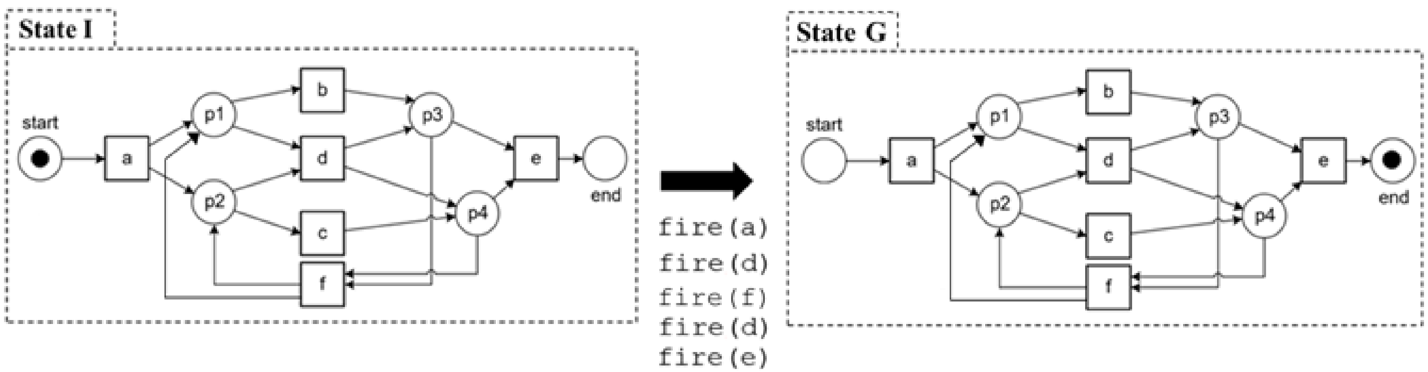
Since S3 is a state satisfying G, the solution found is a **valid plan**.

The Blocks World in PDDL

SubOptimal Plan

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

```
Begin plan
1. (fire a)
2. (fire d)
3. (fire f)
4. (fire d)
5. (fire e)
End plan
```



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

Extensions of PDDL

- **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).

Off-the-Shelf Tools for Planning

Wanna build your own system?

Check out these amazing planning softwares build by the ICAPS community to get started. 🤓

- Fast Downward Planning System
- Tarski An AI Planning Modeling Framework
- ROSPlan | ROS2 Planning and Robotics
- VAL The Plan Validation System
- OPTIC | KCL Planners with time and preferences
- PRP Planner Non-deterministic planning
- Fast Forward Family of satisficing planners
- IBM TOP-K Planners Diverse and Top-Quality planning
- Pyperplan A lightweight STRIPS planner in Python
- LAPKT A lightweight automated planning toolkit
- Planutils A linux-based planning environment
- Planning.Domains Planning on the web
- VS Code | Sublime | Atom PDDL Plugins

Don't see something here? Contribute to the Planning GitHub, add to Planning.Wiki, or send us an email.

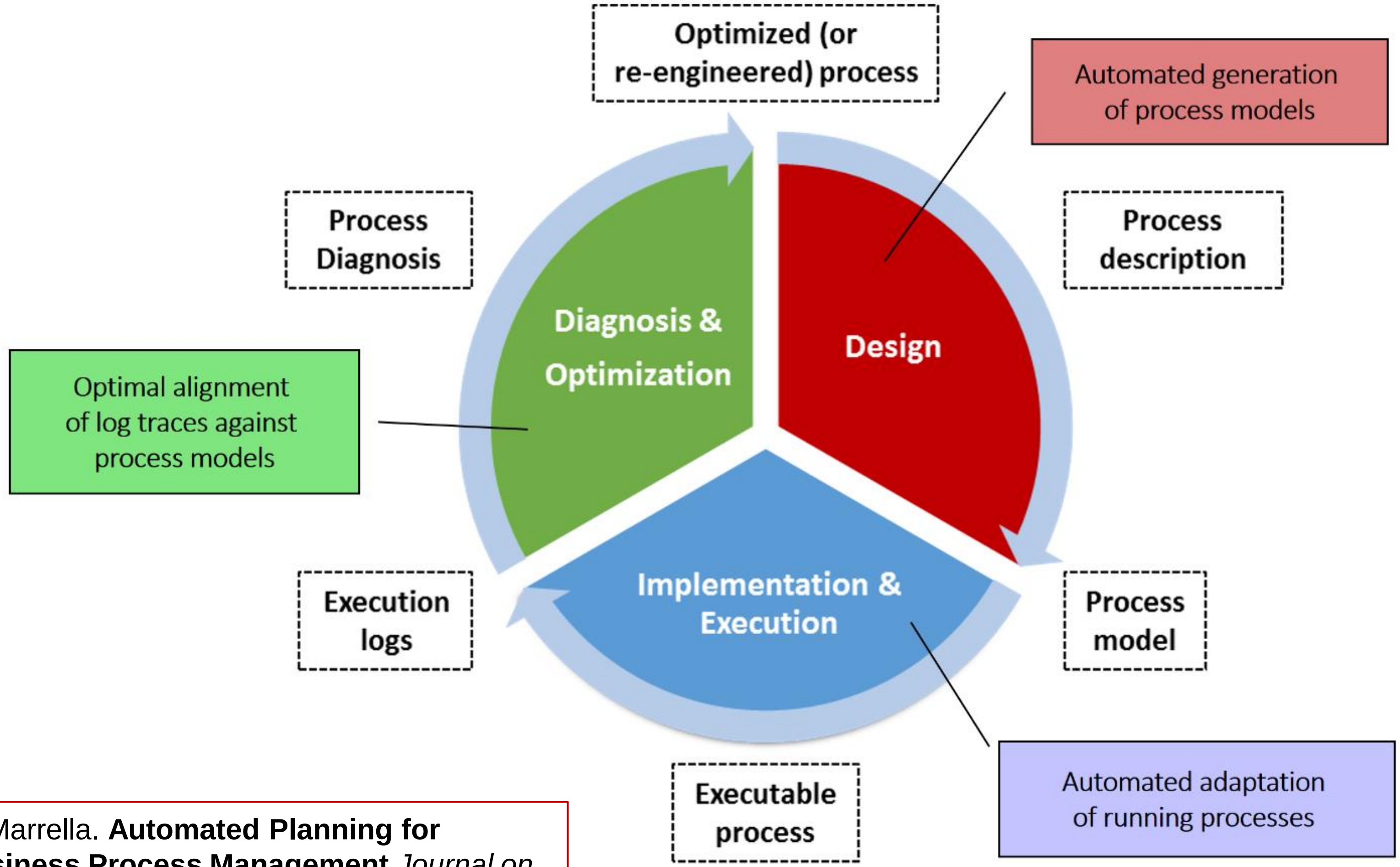
📖📖 Automated Planning
📖 PDDL tips and tricks!

- Link: https://icaps21.icaps-conference.org/demos right hand panel

Syllabus

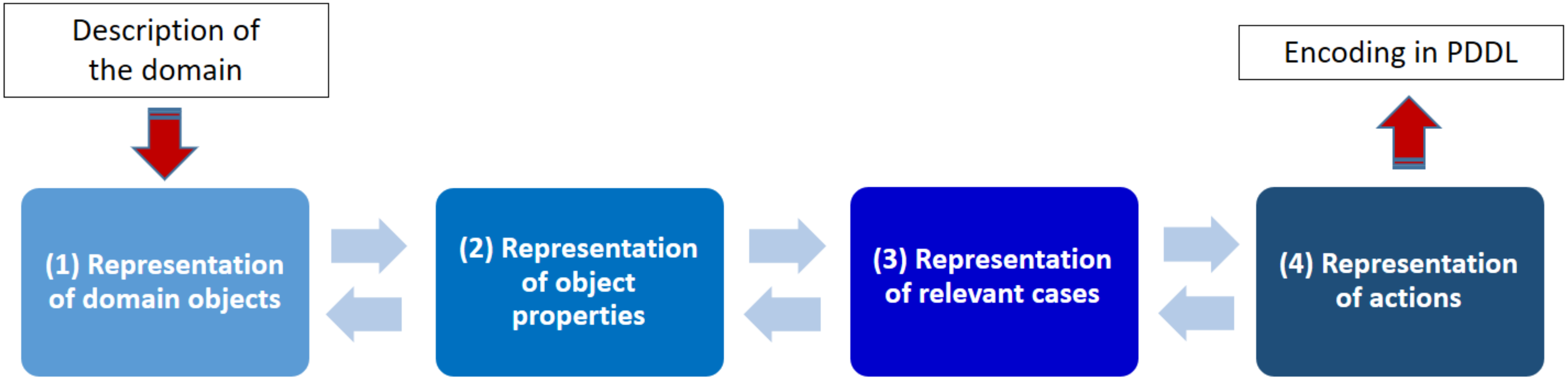
1. Towards AI-Augmented BPM with Planning
2. Basics of Automated Planning
3. **Automated Planning for BPM**
4. Planning-based Declarative Trace Alignment
5. Conclusions

Planning in the life-cycle of BPM



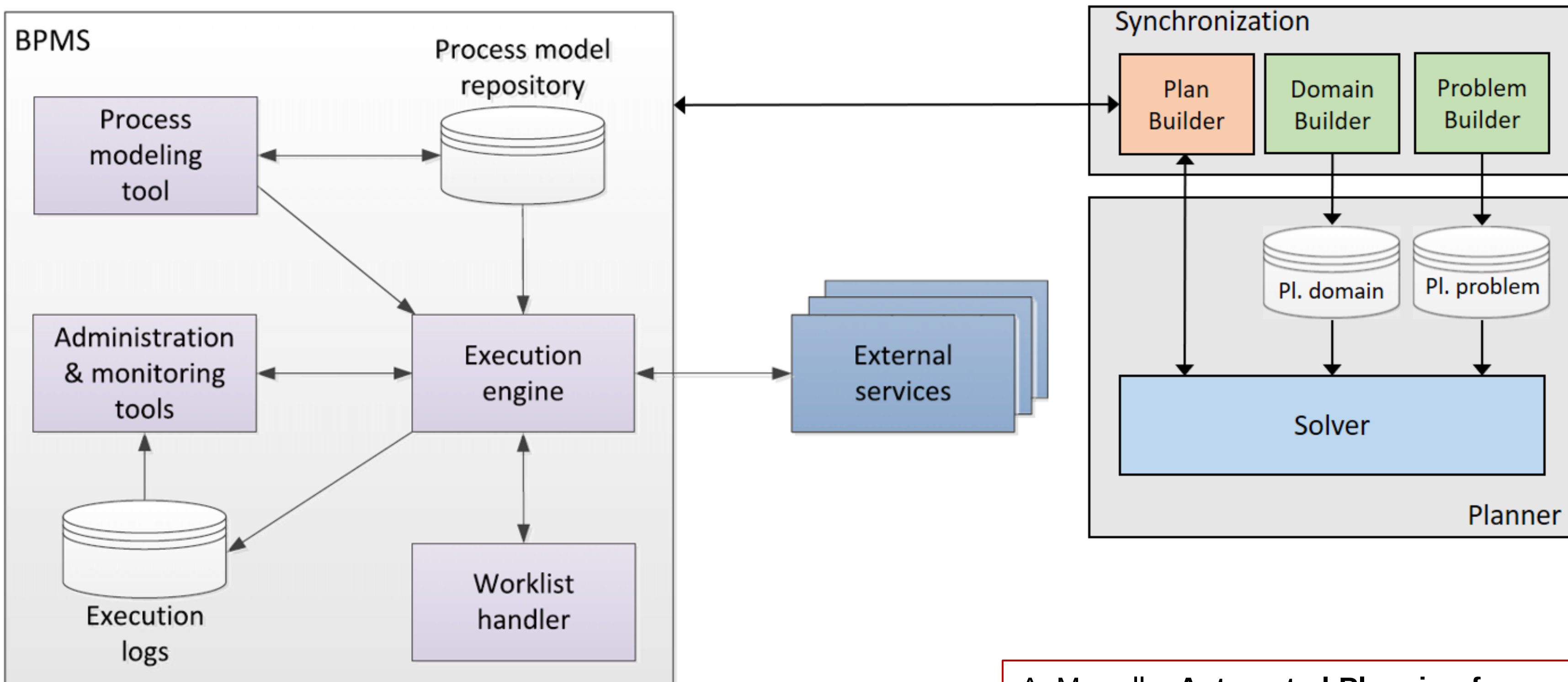
A. Marrella. **Automated Planning for Business Process Management** *Journal on Data Semantics 8 (2), 2019*

A Methodology to build planning problems for BPM



A. Marrella. **Automated Planning for Business Process Management**
Journal on Data Semantics 8 (2), 2019

Integrating planners with BPMS



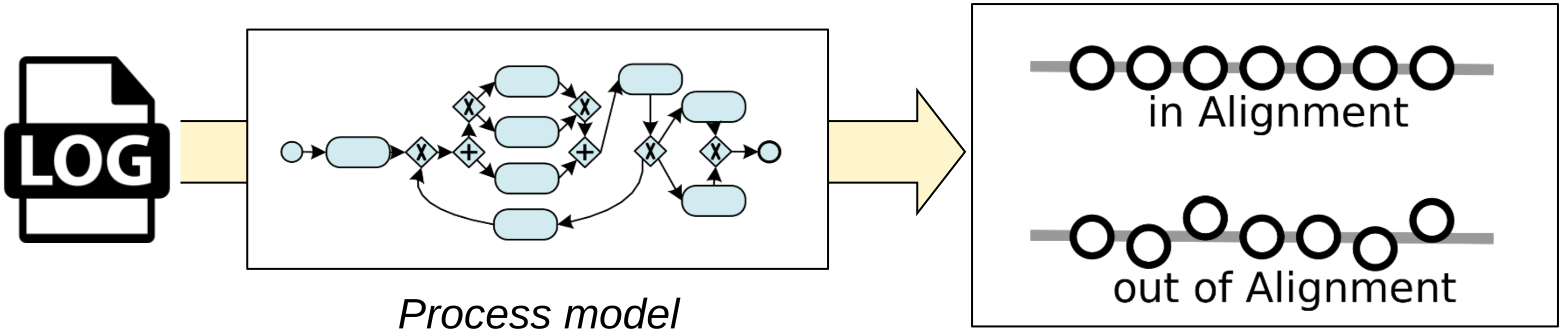
A. Marrella. **Automated Planning for Business Process Management**
Journal on Data Semantics 8 (2), 2019

Syllabus

1. Towards AI-Augmented BPM with Planning
2. Basics of Automated Planning
3. Automated Planning for BPM
4. **Planning-based Declarative Trace Alignment**
5. Conclusions

A successful application
Trace Alignment

- **Process models** are typically not fully enforced by information systems (human behavior is often involved).
  - Traces of execution can be **dirty** with **spurious** or **missing events**.
  - Possible **discrepancies** between the modeled and the observed behavior.



Process model

- **Trace Alignment** finds the **best execution sequence** of a process model (**optimal alignment**) that reproduces an execution trace of the process by pinpointing where it deviates.

Limitations of Trace Alignment

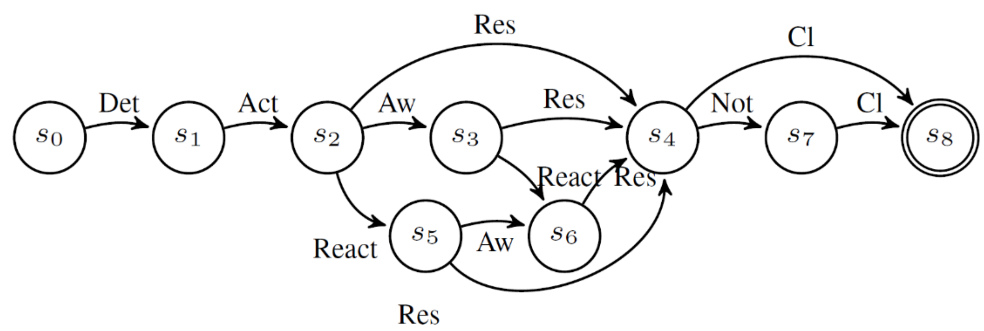
- State-of-the-art solutions to compute optimal alignments in the case of declarative process models:
  - provide **ad-hoc implementations** of the A* algorithm.
  - **do not scale efficiently** when process models and event logs are of considerable size.

G. De Giacomo, F. M. Maggi, A. Marrella, F. Patrizi: **On the Disruptive Effectiveness of Automated Planning for LTLf-Based Trace Alignment**. *Expert systems with applications 82, 2017*

- **Limiting assumption**: alignment algorithms are driven by a **static cost function** assigning fixed costs to deviations.
  - The **context** in which a deviation is found is **neglected**.

M de Leoni, A Marrella, **Aligning real process executions and prescriptive process models through automated planning**. *4th Int. Conf. on Process Mining, ICPM 2022*

Context-aware Trace Alignment



ISO/IEC 27035 – Incident Management process

Det – Detect an incident
Act – Register a ticket
Res – Incident resolution
Aw – Assessment made by 3rd party companies
React – Reactivate the ticket
Not – Resolution notification
Cl – Ticket closure

$\mathbf{t_1}$ = ⟨Act, Aw, Aw, Aw, Res, Cl⟩
$\mathbf{t_2}$ = ⟨Act, Act, Act, Aw, Res, Cl⟩

Alignment

$\mathbf{t_1}$ = ⟨**Det**, Act, Aw, ~~Aw~~, ~~Aw~~, Res, Cl⟩
$\mathbf{t_2}$ = ⟨**Det**, Act, ~~Act~~, ~~Act~~, Aw, Res, Cl⟩

Unitary cost

Cost($\mathbf{t_1}$) = 3
Cost($\mathbf{t_2}$) = 3

Repeated **Aw** → process execution is stuck.
Repeated **Act** → no impact for the security team.

Are equally problematic?
The cost of repeating AW should increase at any occurrence!

SOLUTION: The problem of computing context-aware optimal alignments can be formulated as a **planning problem** in PDDL

Planning-based Trace Alignment

Target: Generate optimal alignments driven by cost models.

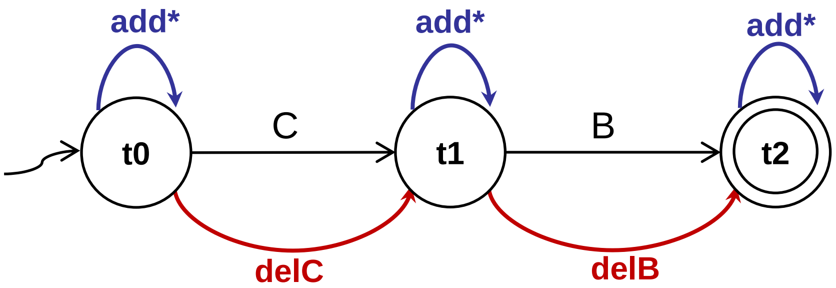
Approach:

1. Process models as **Deterministic Finite State Automata** (**DFAs**)
   - **Clear semantics** to perform formal reasoning over the process model.
   - **Not directly tied** to the prescriptive/declarative nature of the process.
2. DFA-theoretic manipulations to specify the **alignment instructions**.
3. Notion of “context” expressed through a dedicated **cost model**
4. Recasting as a **cost-optimal planning problem** in AI.
5. Automated **planning technology** to find **optimal alignments**.

G. Acitelli, M. Angelini, S. Bonomi, F. M. Maggi, A. Marrella, A. Palma. **Context-Aware Trace Alignment with Automated Planning**. *4th Int. Conf. on Process Mining (ICPM 2022)*

DFA-based solution

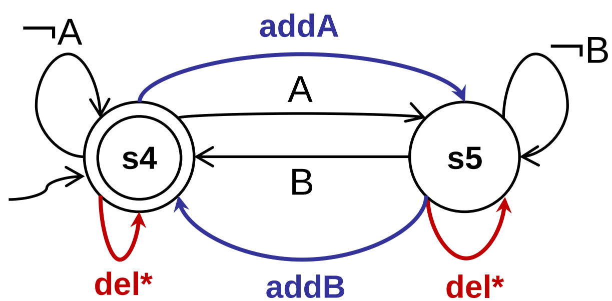
- Trace alignment can be solved using DFAs:
  - One DFA for the trace (***trace automaton***).



  - 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.

DFA-based solution

- One DFA for the process model (***model automaton***) **augmented** to account for adds and dels.



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

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

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 the augmented model automaton.
  - **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 model/trace automaton.

They identify the states of the model 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/model automaton from two states, being `e` the activity involved in the transition.

They hold if `s` is the current/accepting state of a trace/model 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 the model 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 the model 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 C - activity)

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

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

Representation of the trace automaton.

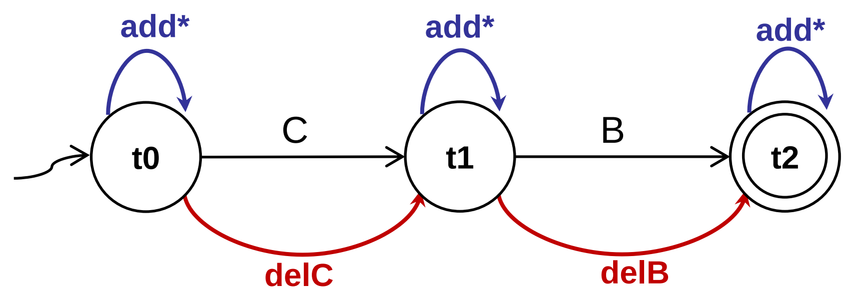
```
(cur_state s4)
(automaton s4 A s5)
(automaton s5 B s4)
(final_state s5))
```

Representation of the model automaton.

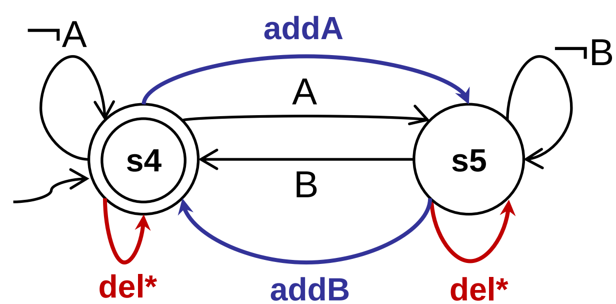
```
(:goal (forall (?s - state)
               (imply (cur_state ?s) (final_state ?s))))

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

Minimization of the total cost of the alignment.

Trace



Model:



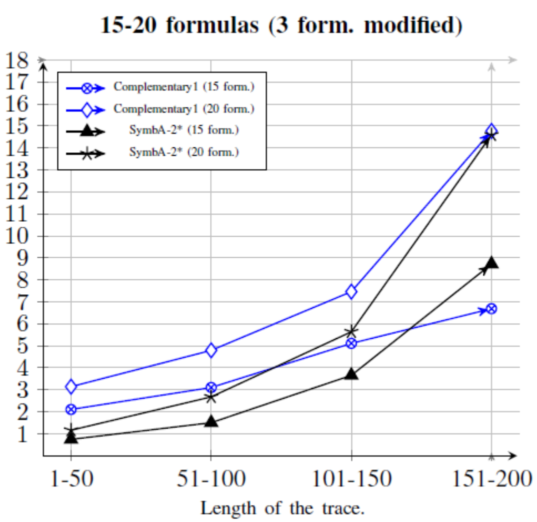
Experiments Results

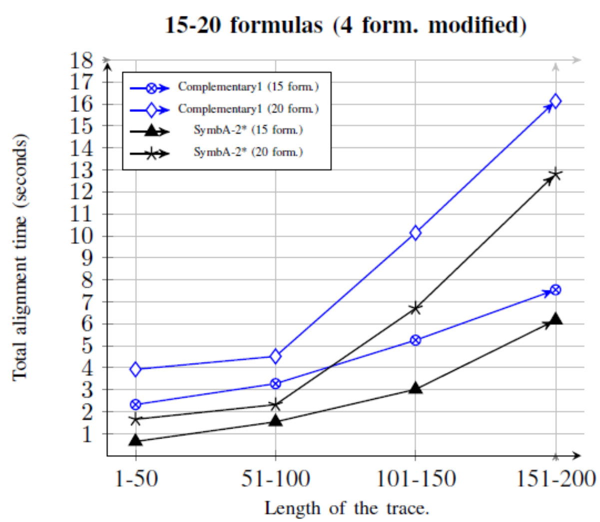
Synthetic logs (DFA with 29182 states & 729526 transitions)

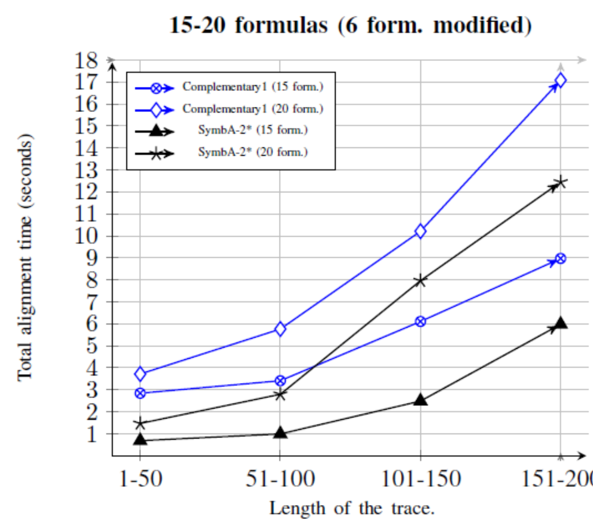
Trace length	SymbA-2* Preprocessing	SymbA-2* Searching	SymbA-2* Steps	Complementary1 Preprocessing	Complementary1 Searching	Complementary1 Steps	Context-Aware Alignment Cost	Alignment Cost
3 form. modified								
1-50	0.24	0.92	51	3.14	$2 \cdot 10^{-3}$	50	1.8	1.8
51-100	0.24	2.44	84	4.79	$4 \cdot 10^{-3}$	84	3.1	2.6
101-150	0.28	5.36	127	7.46	$6 \cdot 10^{-3}$	127	3.7	3.3
151-200	0.37	14.23	183	14.76	$9 \cdot 10^{-3}$	182	4.2	4.2
4 form. modified								
1-50	0.3	1.36	48	3.93	$2 \cdot 10^{-3}$	48	4.8	3.8
51-100	0.24	2.09	77	4.52	$3 \cdot 10^{-3}$	77	6.9	6.9
101-150	0.3	6.41	132	10.13	$7 \cdot 10^{-3}$	132	10.8	10.8
151-200	0.36	12.45	177	16.12	$9 \cdot 10^{-3}$	177	15.9	15.1
6 form. modified								
1-50	0.22	1.25	50	3.72	$2 \cdot 10^{-3}$	50	7.2	6.1
51-100	0.23	2.57	78	5.76	$4 \cdot 10^{-3}$	79	11.2	9
101-150	0.33	7.63	150	10.2	$8 \cdot 10^{-3}$	151	15.6	11.8
151-200	0.35	12.09	184	17.07	$1.1 \cdot 10^{-2}$	185	22.6	17.5

Planners scale well when length of the log traces increases.

Planners do not suffer the presence of noisy logs.







Syllabus

1. Towards AI-Augmented BPM with Planning
2. Basics of Automated Planning
3. Automated Planning for BPM
4. Planning-based Declarative Trace Alignment
5. **Conclusions**

Practical Challenges

- **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

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.
- Planning models are **inherently interpretable** and thus aid human-in-the-loop interfacing to business processes.
- The price for generality is **computational**:
  - Planning over models represented in compact form is **intractable** in the worst case, yet currently large problems can be solved **very quickly**.

FUTURE WORK

- Developing planning systems customized for addressing process mining problems.

Thanks for the attention

Andrea Marrella

marrella@diag.uniroma1.it

Tutorial @ AI4BPM Bridge 2022

February 8, 2023

SAPIENZA
UNIVERSITÀ DI ROMA