Foundations of Framed Autonomy in AI-Augmented BPM Systems

Giuseppe De Giacomo

Department of Computer Science

University of Oxford

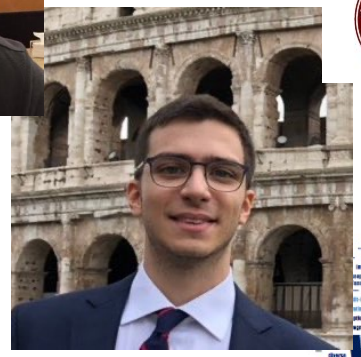
AAAI 2023 Bridge Program on
Artificial Intelligence and Business Process Management
Washington DC, USA, February 8, 2023

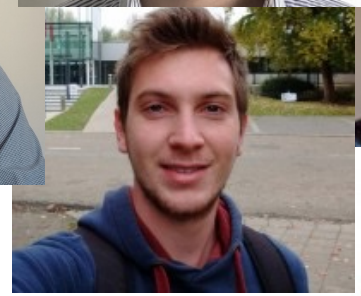
WhiteMech Group

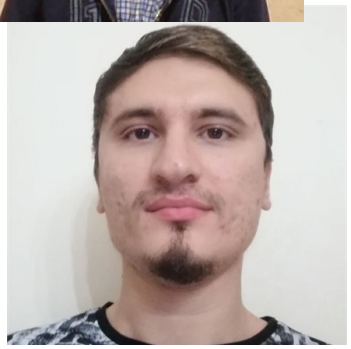
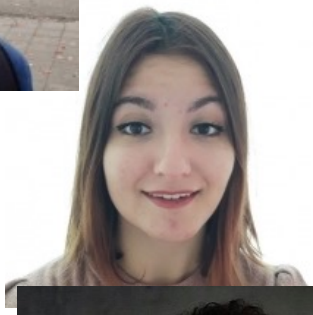
WhiteMech: Whitebox Self Programming Mechanisms

ERC Advanced Grant

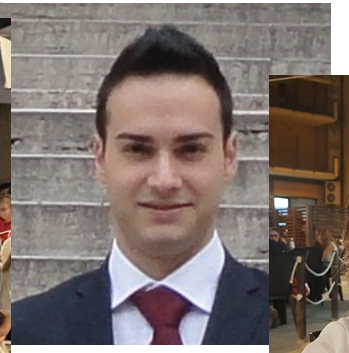






Foundations of Framed Autonomy in AI-Augmented BPM Systems

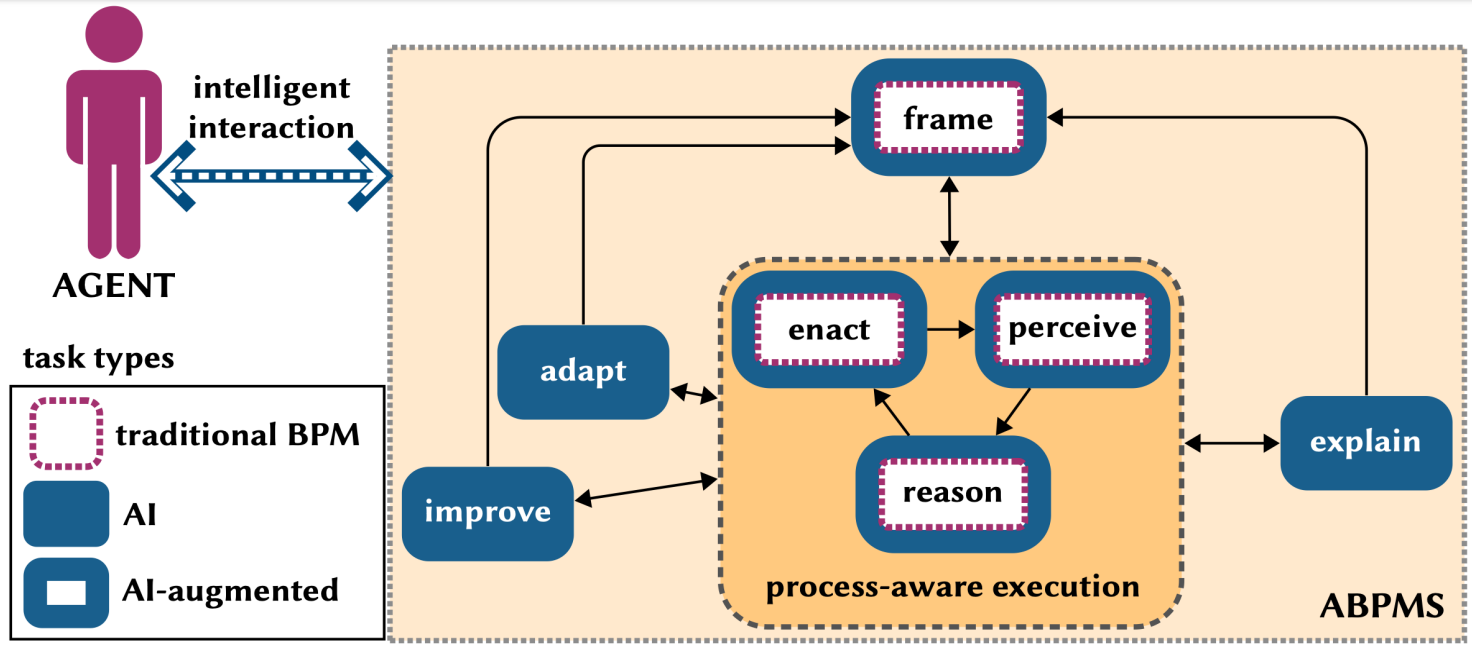
SELF-DELIBERATING PROCESSES

Self-Deliberation in Business Process Management

AI-augmented Business Process Management Systems: A Research Manifesto

Authors: Marlon Dumas, Fabiana Fournier, Lior Limonad, Andrea Marrella, Marco Montali, Jana-Rebecca Rehse, Rafael Accorsi, Diego Calvanese, Giuseppe De Giacomo, Dirk Fahland, Avigdor Gal, Marcello La Rosa, Hagen Völzer, Ingo Weber (Less) Authors Info & Claims

ACM Transactions on Management Information Systems, Volume 14, Issue 1 • March 2023 • Article No.: 11, pp 1–19 • https://doi.org/10.1145/3576047



M. Dumas. Constructing Digital Twins for Accurate and Reliable What-If Business Process Analysis. PROBLEMS'21@BPM'21

Artificial Intelligence is About Building Self-Deliberating Agents!

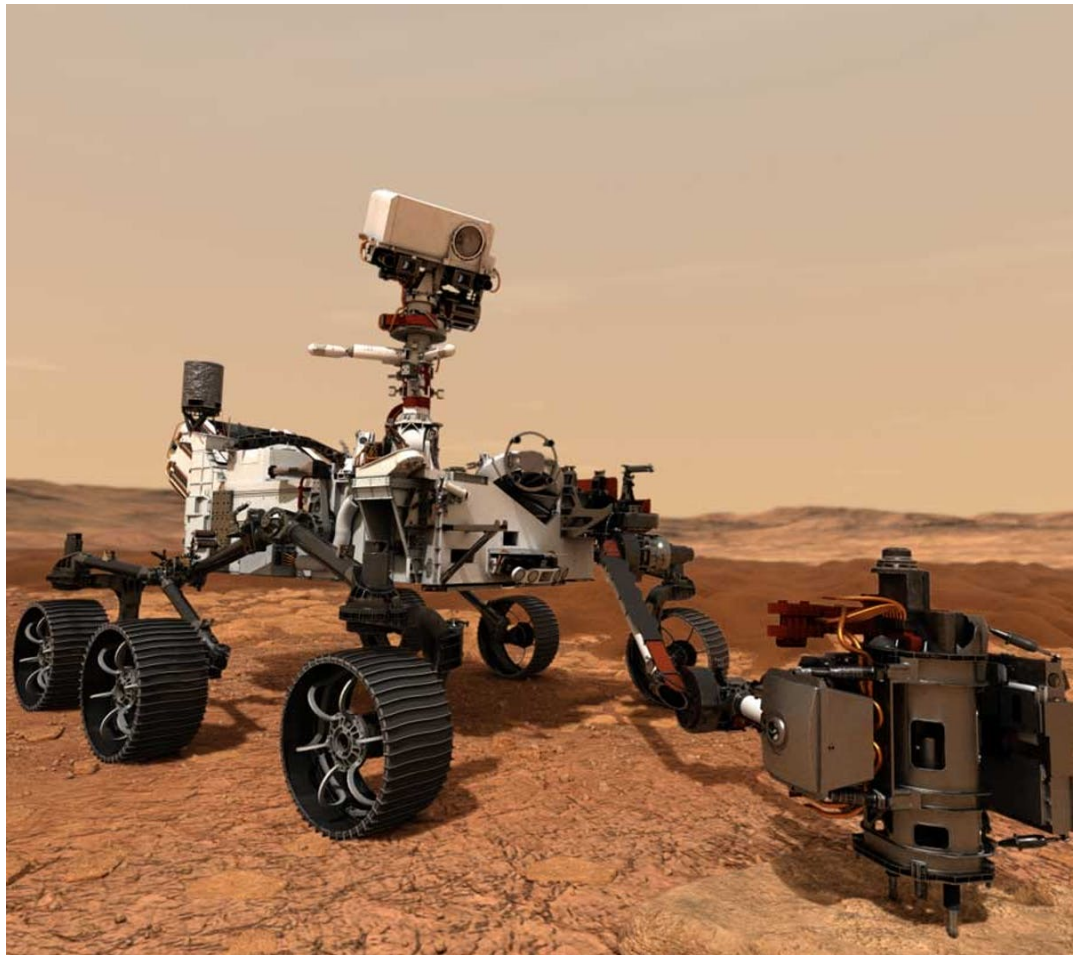
- Autonomy is one of the grand objectives of AI.
- Aims at building autonomous agents/robots that operate in changing, incompletely known, unpredictable environments.
- In other words: Aims at empowering the agent with the ability of
  - deliberating …
  - how to act in the world …
  - autonomously
- First-person view: it is the agent/system that decides what to do

Space Exploration

Delay in communication requires high-level of autonomy during the mission.

Planning and scheduling for temporal extended goals is a top research topic at NASA.

https://mars.nasa.gov/mars2020/

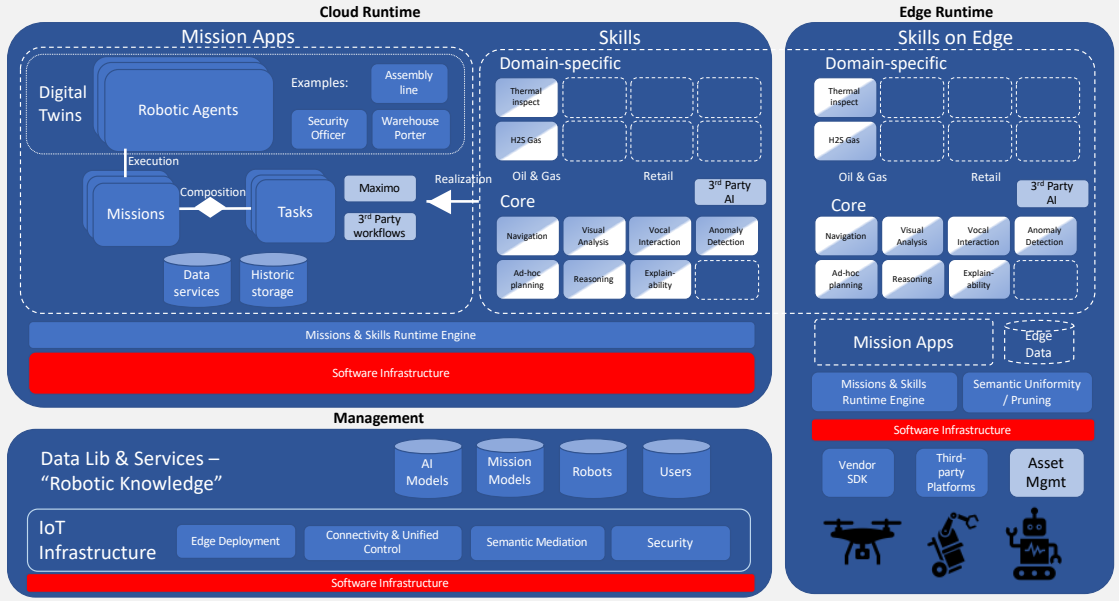


Autonomous Mobile Robots in Logistics

Complex **multi-robot systems** need highly synchronized behaviours to fulfil their job.

These robots need **autonomously resolve unexpected clashes.**

Sophisticated **AMR platforms** under study by industry



Smart Manufacturing, IoT, and Digital Twins

- **Manufacturing as a service:** products to be manufactured are not known in advance and each product may differ from the products manufactured immediately before and immediately after it
- **Internet of Things/Digital Twins** platforms offer infrastructure to deploy embodiment of processes exploiting distributed sensors and actuators
- Analogies with **Service Composition and Orchestration**: synthesize the orchestrator
- **Automated exception handling** is crucial

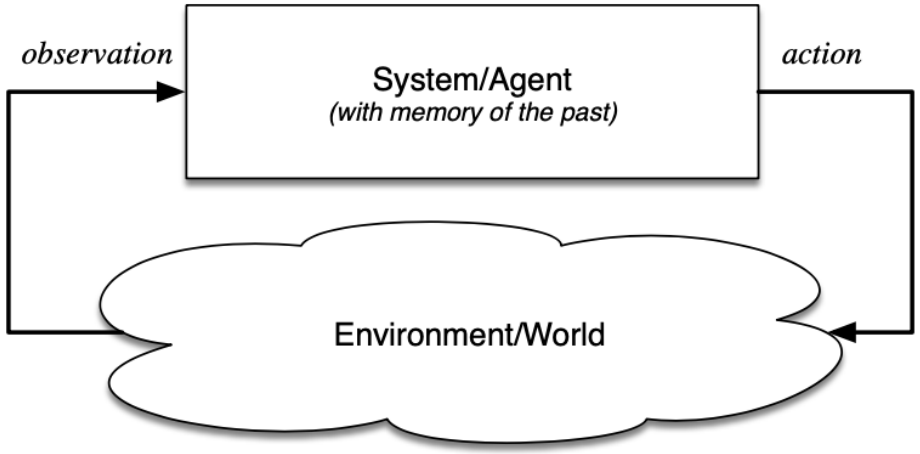
Autonomy Requires Reasoning and Learning

- Autonomy requires:
  - reasoning and planning capabilities
  - learning from experience
- Many areas of AI are concern with autonomy:
  - Logics in AI
  - Knowledge representation and reasoning
  - Planning
  - Multi-agent systems
  - Sequential decision making (MDPs)
  - Reinforcement learning
- Recently: some objectives are shared with automated synthesis in formal methods

WhiteMech: Whitebox Self Programming Mechanisms

ERC Advanced Grant

AI-based Self Deliberating Processes/Dynamic Systems



AI-based Self-Deliberating Dynamic Systems

An AI-based dynamic system in its general form is a function

$$(observation)^* \rightarrow action$$

Note: this is the general form of a process!
[AbadiLamportWolper89]

where

- $(observation)^*$ denotes the history of what observed so far
  (a finite sequence of observations)
- $action$ denotes the next action that the system does

Self-deliberation: such a function is computed/refined/changed while the system/agent is in execution!

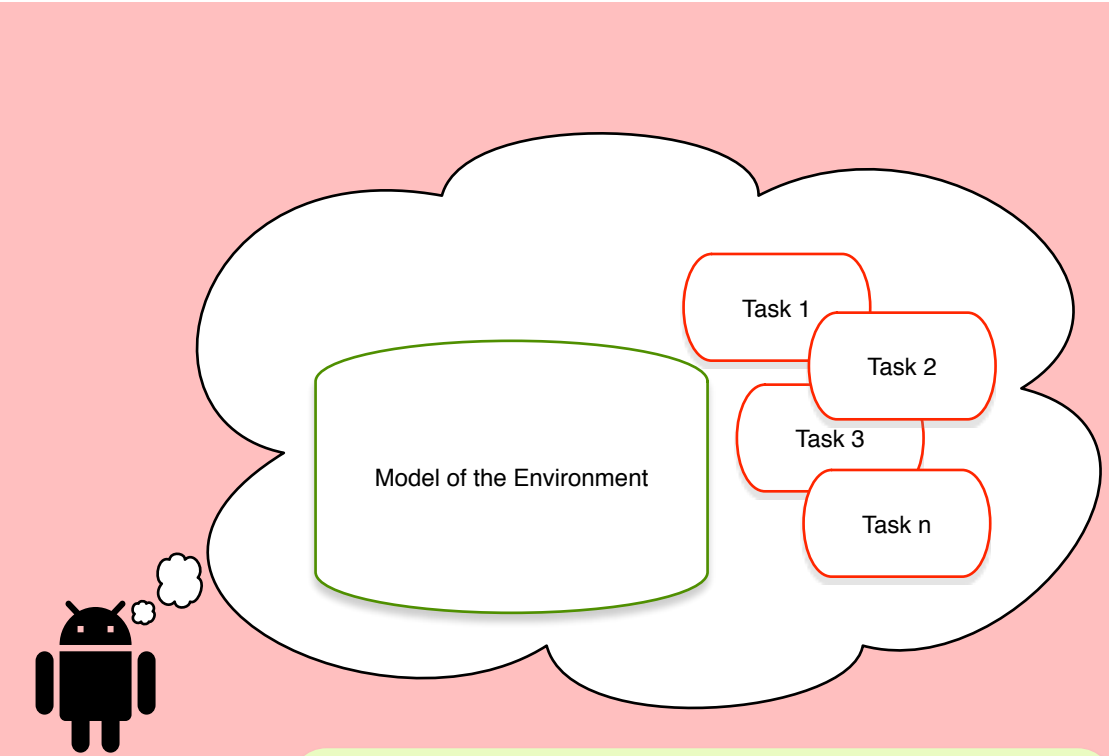
The agent **thinks** *and* **acts**

Foundations of Framed Autonomy in AI-Augmented BPM Systems

PLANNING AND STRATEGIC REASONING

Planning in Deterministic Domains

- **Environment Model (DOM)**
  - Environment model, i.e., the "**domain**" (**DOM**)
  - Specs of environment's **responses to agent's action**
  - **Domain** expressed with specific formalisms
    - STRIPS
    - ADL
    - PDDL
- **DOM** generates a **deterministic transition system**
- **Agent's Task (GOAL)**
  - Spec. of **agent's task**, i.e., the **"goals" (GOALs)**
  - **TASK** expressed as reaching a desired state
- Find agent's **plan/program/strategy/policy** that fulfills **GOAL** in **DOM**



Example: The Yale Shooting Scenario - Deterministic Variant

Example (Yale shooting scenario - deterministic variant)

Consider the following simplified variant of the Yale Shooting Scenario, which has to do with a shooting a turkey named Fred. Find a plan to kill the turkey.

- Fluents:
  - $alive$ - The turkey is alive in the current situation;
  - $loaded$– The gun is loaded in the current situation.
- Actions:
  - $load$ – Load the gun.
    - PRE: Requires that the gun is not loaded;
    - EFF: The gun is loaded
  - $shoot$ - Shoot the gun.
    - PRE: Can always be performed;
    - EFF: If the gun is loaded then it unloads the gun and kills the turkey (not alive), otherwise has no effects
  - $wait$ – A no-op;
    - PRE: Can always be performed;
    - EFF: has no effect on any fluent

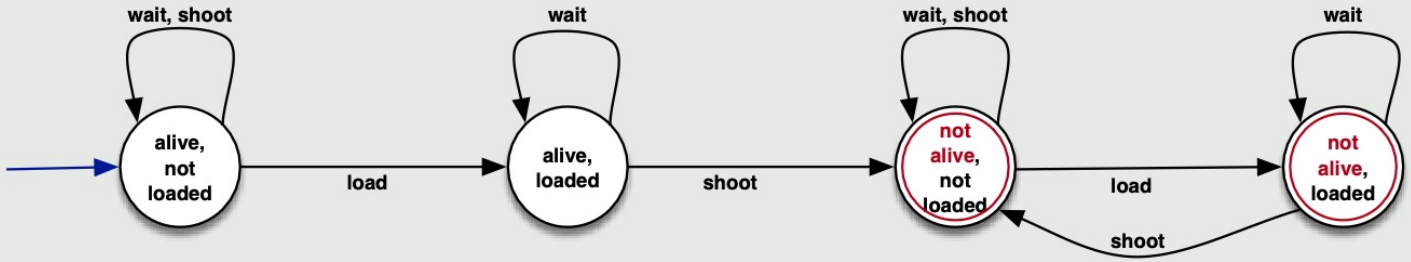
Note: every fluent not mentioned in the effect of actions remains unchanged (frame problem)

- Initial situation description:
  - Initially the turkey is alive, and the gun is not loaded.
- Goal:
  - Kill the turkey, i.e., reach a situation where the turkey is not alive.

Example: The Yale Shooting Scenario - Deterministic Variant

Example (Yale shooting domain with goal)

Consider the goal: $\neg alive$. The domain $\mathcal{D}$ with the goal states highlighted is the following:



Transition system generated by DOM, with GOAL highlighted

Solving Planning in Deterministic Domains

Winning states for deterministic domains

Let denote the **set of goal states** as:

$$[\![G]\!] = \{s \in \mathcal{S} \mid s \models G\}$$

and let's define the **(existential) preimage of a set $\mathcal{E}$** the following function:

$$PreE(\mathcal{E}) = \{s \in \mathcal{S} \mid \exists a \in \alpha(s).\delta(s,a) \in \mathcal{E}\}$$

Compute the set Win of winning states of the domain for goal G, i.e., states from which the agent can reach the goal G, by **least-fixpoint**:

- $Win_0 = [\![G]\!]$ (the goal states)
- $Win_{i+1} = Win_i \cup PreE(Win_i)$
- $Win = \bigcup_i Win_i$

$W_{old} := \emptyset$
$W := [\![G]\!]$
while $(W \neq W_{old})\{$
  $W_{old} := W$
  $W := W \cup PreE(W)$
$\}$
return W

(Computing Win is linear in the number of states in $\mathcal{G}$)

Computing the winning strategy

Let's define $\omega : S \to 2^{\mathcal{A}}$ as:

$$\omega(s) = \{a \in \alpha(s) \mid \text{ if } s \in Win_{i+1} - Win_i \text{ then } \delta(s,a) \in Win_i\}$$

Every way of restricting $\omega(s)$ to return only one action (chosen arbitrarily) gives a **winning strategy** to reach G, i.e., a **plan**.

Universal Plans vs. Classical Plans

- Typically, it is sufficient to have a classical plan (not a universal one), i.e., a sequence of actions that from the initial state leads the agent to a state satisfying the goal.
  - To compute classical plans there are better algorithms (in fact exceptionally fast) – based on heuristic forward search

- The kind of plan just computed is called "**Universal Plan**"

- **Universal Plans** can say what to do to win in every single state!
  - If for any reason during the execution of the plan the agent find itself in an unexpected state (i.e.: the agent expected an effect, but s/he got another one) then the universal plan specifies what to do to win anyway
- **Universal plans** are more **resilient.**

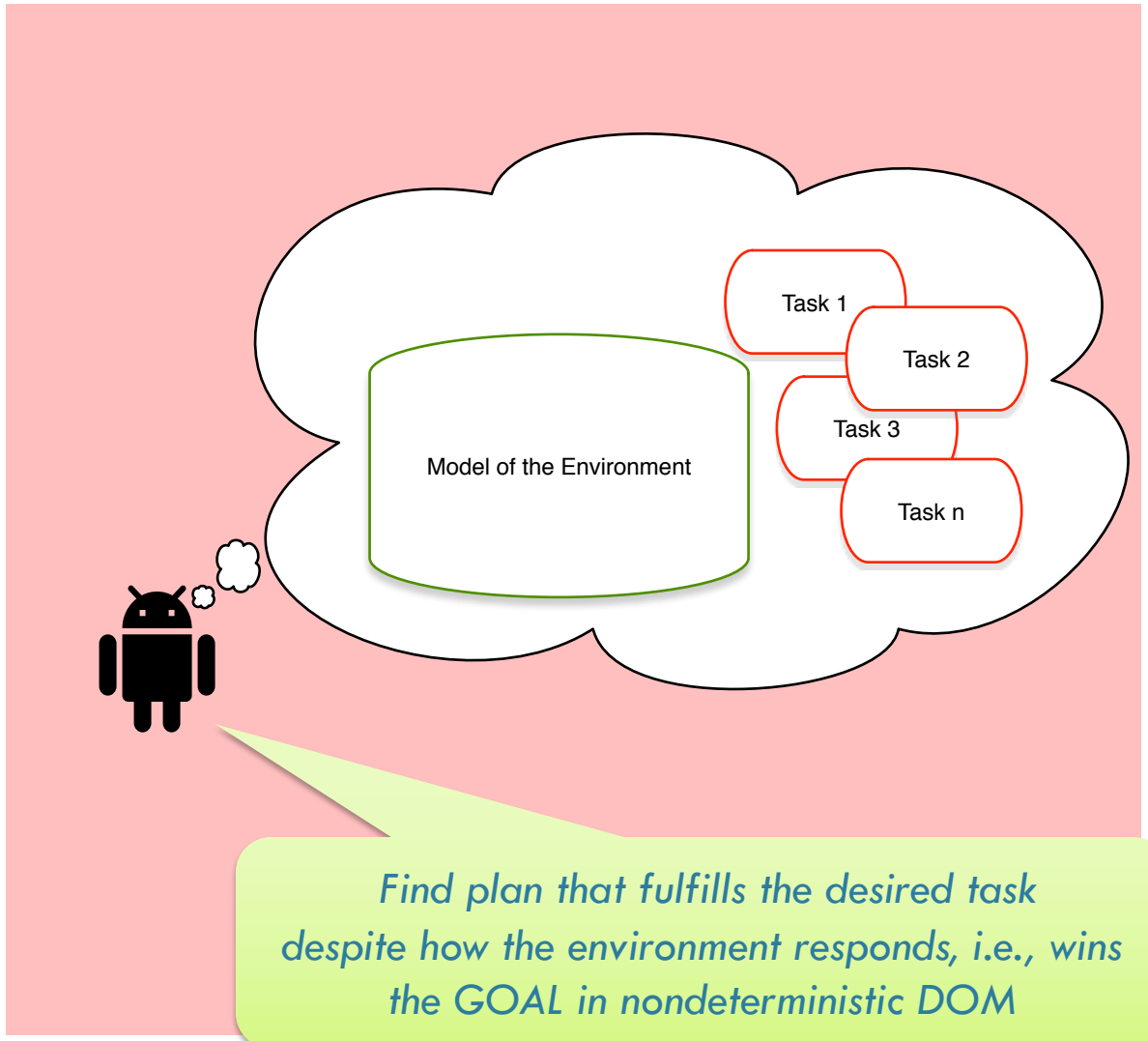
Planning in Nondeterministic Domains

- **Environment Model (DOM)**
  - Environment model, i.e., the "**domain**" (**DOM**)
  - Specs of environment's **responses to agent's action**
  - **Domain** expressed with specific formalisms
    - PDDL – **one of**
  - **DOM** generates a **nondeterministic transition system,** i.e., a

    Game Arena for two players
    - **Agent controlling actions**
    - **Env controlling responses**, i.e., **fluents**

- **Agent's Task (GOAL)**
  - Spec. of **agent's task,** i.e., the **"goals" (GOALs)**
  - **TASK** expressed as reaching a desired state

- Find agent's **plan/program/strategy/policy** that fulfills **GOAL** in **DOM**



Example: The Yale Shooting Scenario - Nondeterministic Variant

Example (Yale shooting scenario – nondeterministic variant)

Consider the following nondeterministic variant of the Yale Shooting Scenario.

- Fluents:
  - $alive$ - The turkey is alive in the current situation;
  - $loaded$– The gun is loaded in the current situation;
  - $jammed$– The gun is jammed in the current situation.
- Actions:
  - $load$ – Load the gun.
    - PRE: Requires that the gun is not loaded;
    - EFF: It has two alternative possible effects:
      Either (i) it loads the gun and does not jam it; or (ii) it loads the gun and jams it.

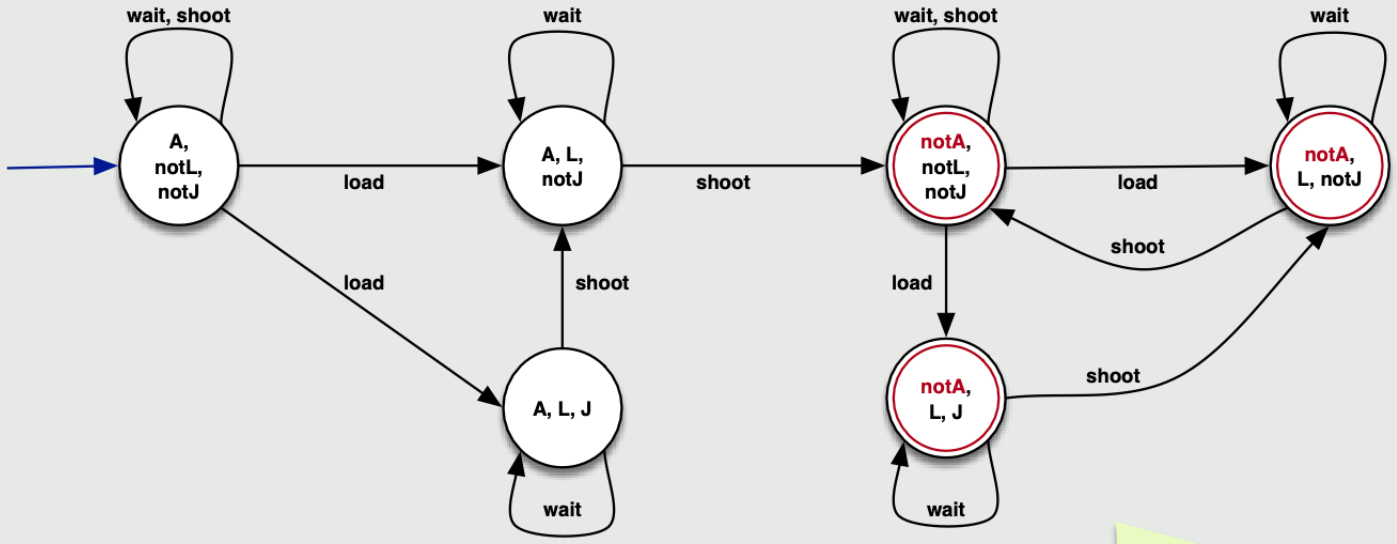
    (Use PDDL "oneof" for expressing alternative effects.)
  - $shoot$ - Shoot the gun.
    - PRE: Can always be performed;
    - EFF: If the gun is loaded and not jammed then it unloads the gun and kills the turkey (not alive), if the gun is loaded and jammed then it unjams the gun; if the gun is not loaded then it does nothing.
  - $wait$ – A no-op;
    - PRE: Can always be performed;
    - EFF: has no effect on any fluent

  Note: every fluent not mentioned in the effect of actions remains unchanged (frame problem)
- Initial situation description:
  - Initially the turkey is alive, the gun is not loaded, and the gun is not jammed.
- Goal:
  - Kill the turkey, i.e., reach a situation where the turkey is not alive.

Example: The Yale Shooting Scenario - Nondeterministic Variant

Example (Yale shooting domain with goal)

Consider the goal: $\neg alive$. The domain $\mathcal{D}$ with the goal states highlighted is the following:



Transition system generated by DOM is an arena where a game is played,

- *By **AGENT** controlling the **actions** and*
- *By **ENV** controlling the response (the **fluents**)*

Solving Planning in Nondeterministic Domains

Winning states for nondeterministic domains

Let denote the **set of goal states** as:

$$[\![G]\!] = \{s \in \mathcal{S} \mid s \models G\}$$

and let's define the **(adversarial preimage of a set $\mathcal{E}$** the following function:

$$PreAdv(\mathcal{E}) = \{s \in \mathcal{S} \mid \exists a \in \alpha(s).\forall s' \in \mathcal{S}.\, \delta(s,a,s') \supset s' \in \mathcal{E}\}$$

Compute the set Win of winning states of the domain for goal G, i.e., states from which the agent can reach the goal G, by **least-fixpoint**:

- $Win_0 = [\![G]\!]$ (the goal states)
- $Win_{i+1} = Win_i \cup PreAdv(Win_i)$
- $Win = \bigcup_i Win_i$

$W_{old} := \emptyset$
$W := [\![G]\!]$
while $(W \neq W_{old})\{$
  $W_{old} := W$
  $W := W \cup PreAdv(W)$
$\}$
return W

(Computing Win is linear in the number of states in $\mathcal{D}$)

Computing the winning strategy

Let's define $\omega : S \to 2^{\mathcal{A}}$ as:

$$\omega(s) = \{a \in \alpha(s) \mid \text{ if } s \in Win_{i+1} - Win_i \text{ then } \forall s'.\delta(s,a,s') \supset s' \in Win_i\}$$

Every way of restricting $\omega(s)$ to return only one action (chosen arbitrarily) gives a **winning strategy** to reach G, i.e., a **plan**.

Foundations of Framed Autonomy in AI-Augmented BPM Systems

LINEAR TIME LOGICS ON FINITE TRACES

Formal Methods

- Rigorous guarantees about the behavior of computational systems
- Wide-spread industrial adoption
- Main tasks
  - **Formalisms/Logics for specs of dynamic properties:**
    - E.g., Linear Temporal Logic
  - **Verification:**
    - Check if the system satisfies specs.
  - **Synthesis:**
    - Synthesize a system that satisfies specs.

CS/AI & Logics

The connection between Logic and Computer Science has been widely acknowledged by the academic community, e.g. in:

- Computational complexity
- Database theory
- Programming languages
- Knowledge representation
- Automated reasoning
- Computer-aided verification
- Formal methods
- …

CS/AI & Temporal Logics

Temporal Logics are useful to reason about propositions through time.

Applications in CS/AI:

- Formal verification of programs
- Temporal synthesis
- Planning with temporal goals
- Markov Decision Processes
- Business Process Management

Many techniques based on temporal logics rely on the connection with **finite automata**.

Infinite-trace vs Finite-trace semantics

The “big bang” of the application of temporal logic to program verification: **Linear Temporal Logic** (LTL) (Pnueli, 1977)

Semantics of LTL is over ω-words, i.e. infinite traces

Note:

- ω-automata algorithms scale well as long as you do not have to determinize ☺
- For synthesizing strategies/policy determinization is essential ☹
- In AI are more interested in finite-trace semantics ☺

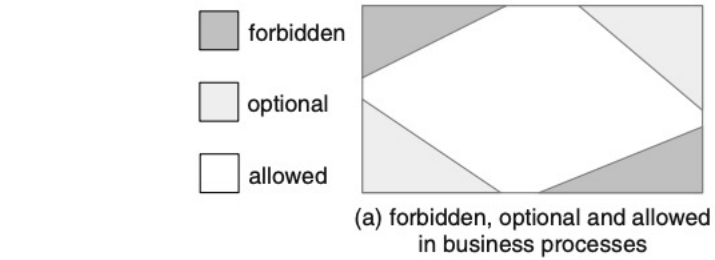
Declarative Processes

Late 2000's: The BPM community comes up with a brilliant idea:
[PesicVanDerAalst06] [AlbertiEtAlt06] [Montali2010] [PesicBovsnavkiVanDerAalst10]

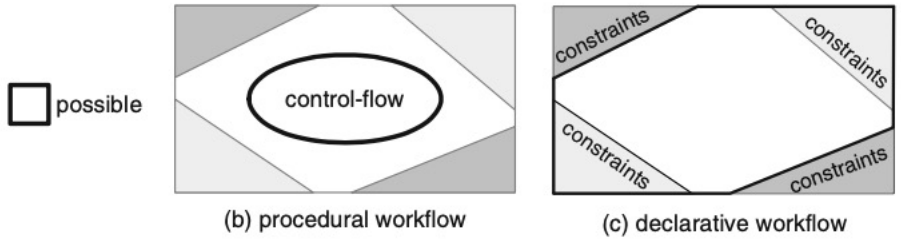
The idea:
- Give the rules that a process should satisfy
- And nothing else!
- Extract the process from the rules only

Which specs? [PesicVanDerAalst06]
- The one most used in formal methods for specifying process properties
- Linear Time Logic (LTL)

In other words:
- Drop explicit representation of process, and
- Use instead LTL formulas to specify the allowed process traces



(a) forbidden, optional and allowed in business processes



(b) procedural workflow

(c) declarative workflow

Declarative Processes

Late 2000's: The BPM community comes up with a brilliant idea:
[PesicVanDerAalst06] [AlbertiEtAlt06] [Montali2010] [PesicBovsnavkiVanDerAalst10]

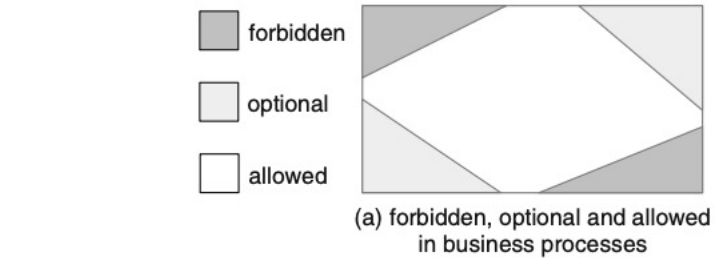
The idea:

- Give the rules that a process should satisfy
- And nothing else!
- Extract the process from the rules only

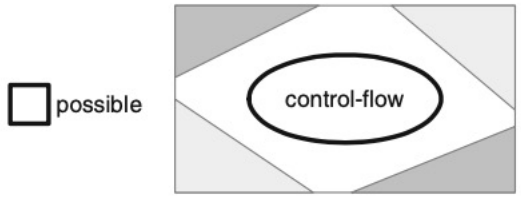
In other words: Automatically synthesize process from declarative specs

It can be seen as the fulfillment of the CS dream:

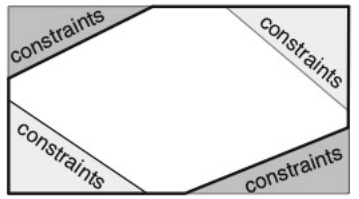
- Devise a technique for the "mechanical translation of human-understandable task specifications to a program that is known to meet the specifications." [Vardi - The Siren Song of Temporal Synthesis 2018] (more later)



(a) forbidden, optional and allowed in business processes



(b) procedural workflow



(c) declarative workflow

Declarative Processes

Originally a controlled set of notable LTL formulas on were proposed for process specification (and a suitable graphical notation provided) [PesicVanDerAalst06]

Example (Main DECLARE Patterns)

NAME	NOTATION	LTL_f	DESCRIPTION
Existence	1..* a	$\Diamond a$	a must be executed at least once
Resp. existence	a ● — b	$\Diamond a \supset \Diamond b$	If a is executed, then b must be executed as well
Response	a ● → b	$\Box(a \supset \Diamond b)$	Every time a is executed, b must be executed afterwards
Precedence	a →● b	$\neg b \mathcal{W} a$	b can be executed only if a has been executed before
Alt. Response	a ●⟹ b	$\Box(a \supset \bigcirc(\neg a \mathcal{U} b))$	Every a must be followed by b, without any other a inbetween
Chain Response	a ●⟹ b	$\Box(a \supset \bigcirc b)$	If a is executed then b must be executed next
Chain Precedence	a ⟹● b	$\Box(\bigcirc b \supset a)$	Task b can be executed only immediately after a
Not Coexistence	a ●‖ b	$\neg(\Diamond a \wedge \Diamond b)$	Only one among tasks a and b can be executed
Neg. Succession	a ●‖→ b	$\Box(a \supset \neg\Diamond b)$	Task a cannot be followed by b, and b cannot be preceded by a
Neg. Chain Succ.	a ‖⟹● b	$\Box(a \supset \bigcirc\neg b)$	Tasks a and b cannot be executed next to each other

Assumes only one activity (proposition) true at each point in time.

Declarative Processes

Originally a controlled set of notable LTL formulas on were proposed for process specification (and a suitable graphical notation provided) [PesicVanDerAalst06]

Can we use any LTL formula as a declarative spec?

No! What if the spec is: ***always eventually Happy*** *?*

Yes, if you focus on finite trace! *(Specs in LTLf instead of LTL)*

Example (Main DECLARE Patterns)

NAME	NOTATION	LTL_f	DESCRIPTION
Existence	1..* a	$\Diamond a$	a must be executed at least once
Resp. existence	a — b	$\Diamond a \supset \Diamond b$	If a is executed, then b must be executed as well
Response	a → b	$\Box(a \supset \Diamond b)$	Every time a is executed, b must be executed afterwards
Precedence	a ⇢ b	$\neg b \mathcal{W} a$	b can be executed only if a has been executed before
Alt. Response	a ⇢ b	$\Box(a \supset \bigcirc(\neg a \mathcal{U} b))$	Every a must be followed by b, without any other a in
Chain Response	a ⇒ b	$\Box(a \supset \bigcirc b)$	If a is executed then b must be executed next
Chain Precedence	a ⇒ b	$\Box(\bigcirc b \supset a)$	Task b can be executed only immediately after a
Not Coexistence	a ⇸ b	$\neg(\Diamond a \wedge \Diamond b)$	Only one among tasks a and b can be executed
Neg. Succession	a ⇸ b	$\Box(a \supset \neg\Diamond b)$	Task a cannot be followed by b, and b cannot be preceded by a
Neg. Chain Succ.	a ⇏ b	$\Box(a \supset \bigcirc\neg b)$	Tasks a and b cannot be executed next to each other

Assumes only one activity (proposition) true at each

G. De Giacomo, R. De Masellis, M. Montali: Reasoning on LTL on Finite Traces: Insensitivity to Infiniteness. AAAI 2014

Focus on Finite Traces is Shared by AI

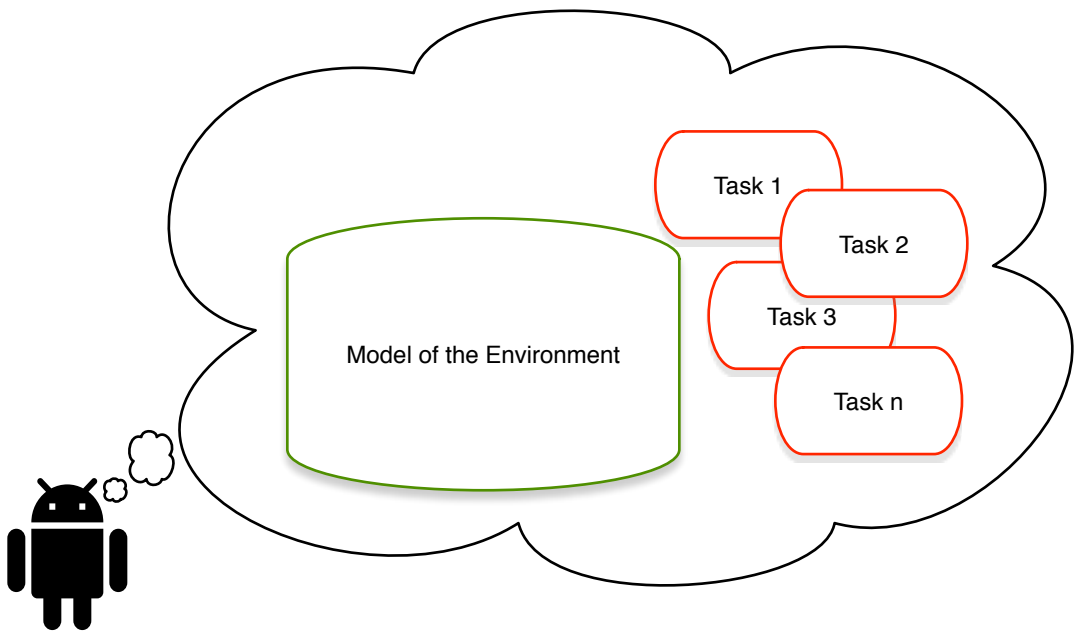
Planning in AI:

- Is all about having a task specification or "goal" and producing a "plan" (or strategy or policy) to satisfy the task in the environment model.

- Which tasks?
  - A **task that terminates**!
  - Typically, just reaching a certain state in the environment

Why tasks that terminates?

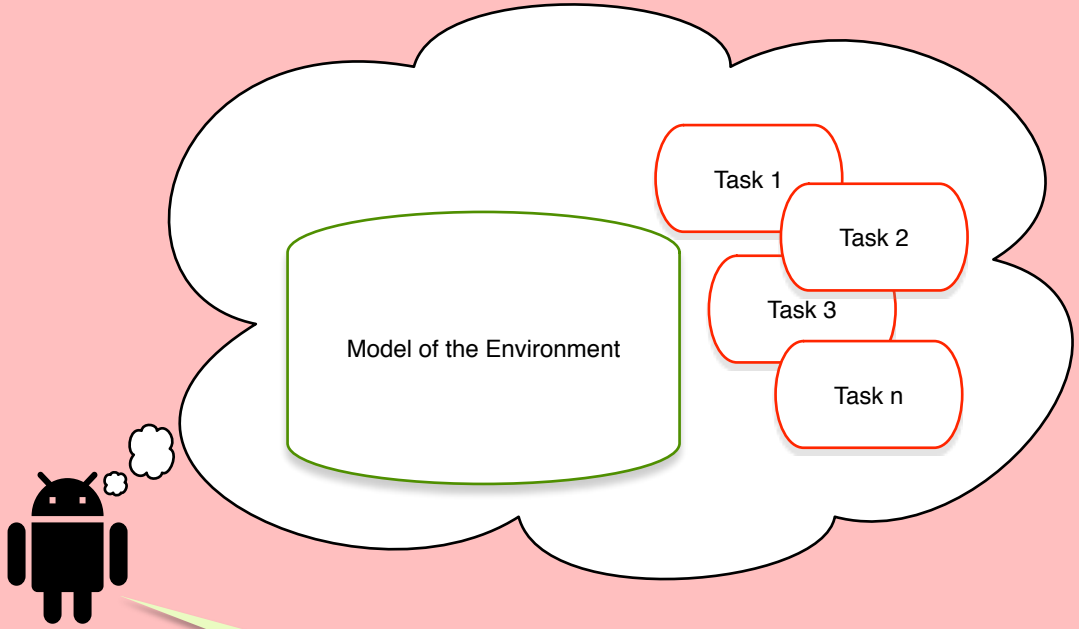
- Because it is the agent that is planning/reasoning
- If the task would not terminate, the agent would be stuck into doing the same task forever
- But then, why bother with equipping it with a model of the environment and of the task at all?
- Note it is the agent, NOT the designer, who has such a model



Specify Models and Tasks as Programs in Formal Methods

- **Environment Model (ENV)**
  - Spec. of **environment** possible behaviors **(ENV)**
  - Think of each **behavior** as a choice function resolving nondeterminism of a nondeterministic domain
  - **ENV** expressed as
    - **nondeterministic planning domains**
    - **LTL/LTLf** specifications
- **Agent's Task (TASK/GOAL)**
  - Spec. of **agent's task**
  - **TASK** expressed in **LTL/LTLf**
- **Find agent's** plan/behavior/policy/**strategy** that fulfills **TASK** against **all** behaviors of **ENV**

*Specify **ENV** and **TASK** as with formalism used in **Formal Methods** e.g., **LTL/LTLf***



*Find agent's **strategy** that is **winning against ENV***

Linear Time Logic on Finite Traces

LTL_f

$$\varphi ::= A \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \bigcirc\varphi \mid \varphi_1 \,\mathcal{U}\, \varphi_2 \mid \bullet\varphi \mid \Diamond\varphi \mid \Box\varphi \mid \mathit{Last}$$

- A: atomic propositions
- $\neg\varphi$, $\varphi_1 \wedge \varphi_2$: boolean connectives
- $\bigcirc\varphi$: "next step exists and at next step (of the trace) φ holds"
- $\varphi_1 \,\mathcal{U}\, \varphi_2$:"eventually φ_2 holds, and φ_1 holds until φ_2 does"
- $\bullet\varphi \doteq \neg\bigcirc\neg\varphi$: "if next step exists then at next step φ holds" *(weak next)*
- $\Diamond\varphi \doteq \mathtt{true}\,\mathcal{U}\,\varphi$: "$\varphi$ will eventually hold"
- $\Box\varphi \doteq \neg\Diamond\neg\varphi$: "from current till last instant φ will always hold"
- $\mathit{Last} \doteq \neg\bigcirc\mathtt{true}$: denotes last instant of trace.

Main formal properties:

- **Expressibility:** FOL over finite sequences or Star-free RE
- **Reasoning:** satisfiability, validity, entailment PSPACE-complete
- **Model Checking:** linear on TS, PSPACE-complete on formula

Linear Time Logic on Finite Traces

Examples

$\Diamond A$	"eventually A"	*reachability*
$\Box A$	"always A"	*safety*
$\Box(A \supset \Diamond B)$	"always if A then eventually B"	*reactiveness*
$A \mathcal{U} B$	"A until B"	*strong until* – *stronger than English until*
$A \mathcal{U} B \vee \Box A$	"A until B"	*weak until* – *just like English until*

We can use plain English words instead of symbols!

eventually A	"eventually A"	*reachability*
always A	"always A"	*safety*
always$(A \supset$ *eventually* $B)$	"always if A then eventually B"	*reactiveness*
A *until* B	"A until B"	*strong until* – *stronger than English until*
A *until* $B \vee$ *always* A	"A until B"	*weak until* – *just like English until*

Linear Dynamic Logic on Finite Traces

LDL$_f$

$$\varphi ::= \phi \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \langle\rho\rangle\varphi \mid [\rho]\varphi \qquad \rho ::= \phi \mid \varphi? \mid \rho_1 + \rho_2 \mid \rho_1;\rho_2 \mid \rho^*$$

- ϕ: propositional formula on current state/instant
- $\neg\varphi$, $\varphi_1 \wedge \varphi_2$: boolean connectives
- ρ is a regular expression on propositional formulas
- $\langle\rho\rangle\varphi$: exists an "execution" of RE ρ that ends with φ holding
- $[\rho]\varphi$: all "executions" of RE ρ (along the trace!) end with φ holding

In the infinite trace setting, such enhancement strongly advocated by industrial model checking [ForSpec, PSL].

Main formal properties:

- **Expressibility:** MSO over finite sequences: adds the power of recursion (as RE)
- **Reasoning:** satisfiability, validity, entailment PSPACE-complete
- **Model Checking:** linear on TS, PSPACE-complete on formula

LTLf and Automata

Key point

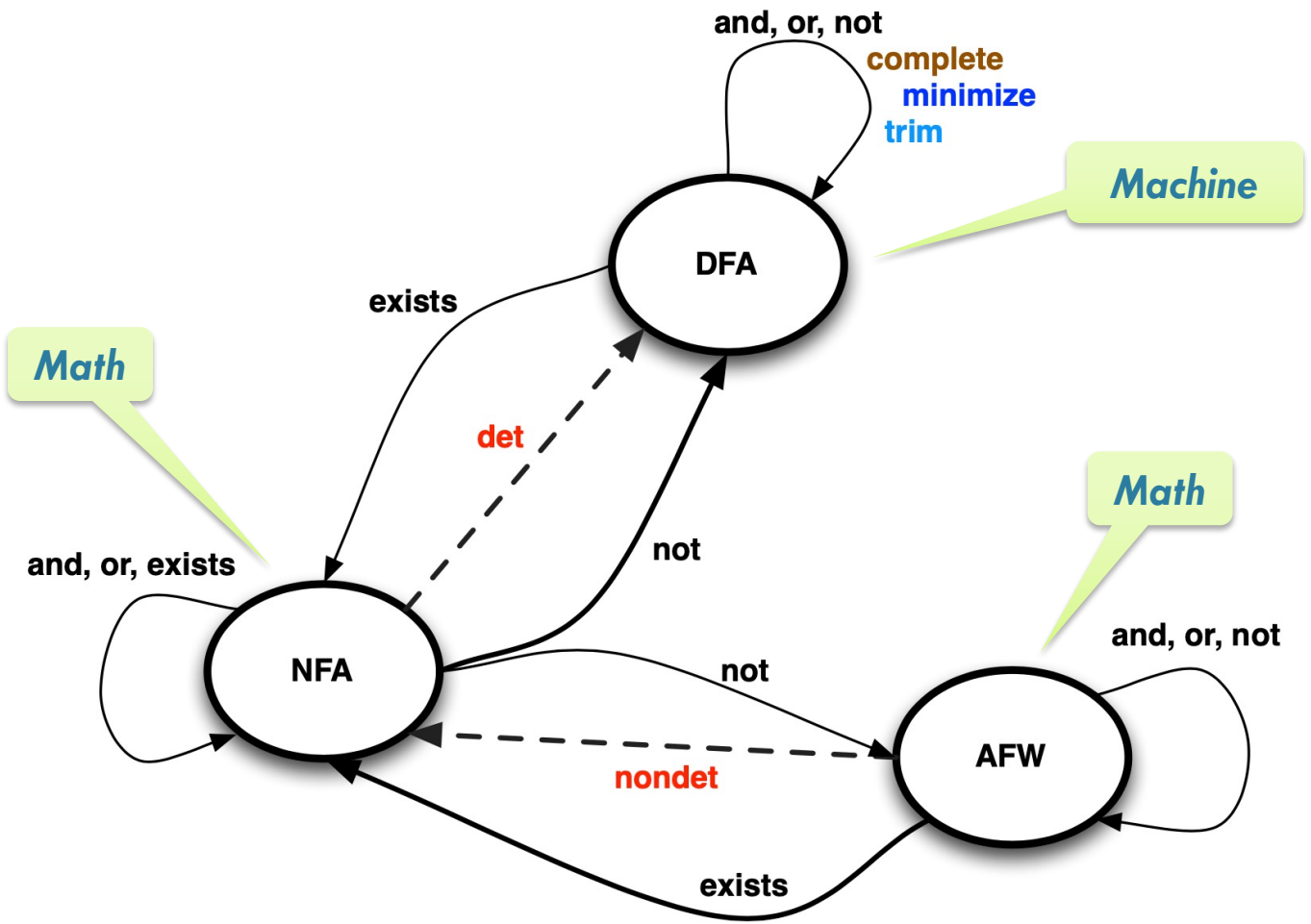
$\text{LTL}_f/\text{LDL}_f$ formulas can be translated into a finite-state automaton on finite words $\mathcal{A}_\varphi$ such that:

$$t \models \varphi \text{ iff } t \in \mathcal{L}(\mathcal{A}_\varphi)$$

- in linear time if $\mathcal{A}_\varphi$ is an Alternating Automata (AFW);
- in exponential time if $\mathcal{A}_\varphi$ is an Nondeterministic Finite-state Automaton (NFA);
- in double exponential time if $\mathcal{A}_\varphi$ is an Deterministic Finite-state Automaton (DFA).

We can compile reasoning into automata based procedures!

Regular Automata: a Roadmap



LTLf to DFA

Key point

$\text{LTL}_f/\text{LDL}_f$ formulas can be translated into deterministic finite state automata (DFA).

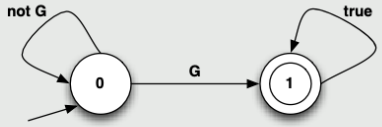
$$t \models \varphi \text{ iff } t \in \mathcal{L}(A_\varphi)$$

where A_φ is the DFA φ is translated into.

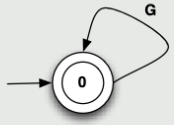
NB: DFA canical after minimization!

Example (Automata for some $\text{LTL}_f/\text{LDL}_f$ formulas)

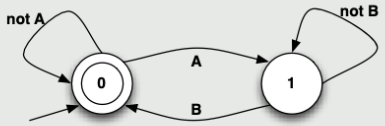
$\Diamond G$



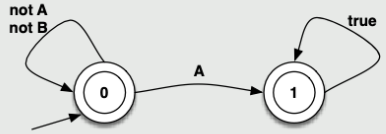
$\Box G$



$\Box(A \supset \bigcirc\Diamond B)$

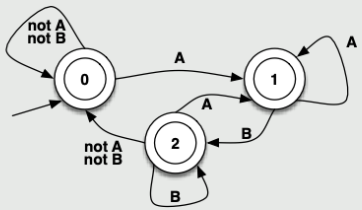


$\neg B \,\mathcal{U}\, A \vee \Box\neg B$ *"A before B"*



$\langle(\neg B^*; A; \neg B^*; B; B)^*; \neg B^*\rangle end$

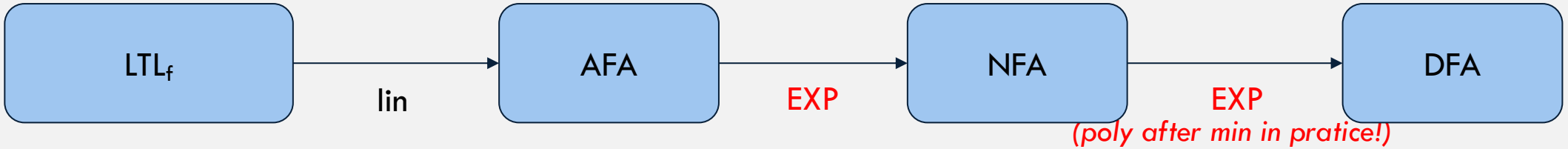
"each time new A before B" (*A and B not true simultaneously*)



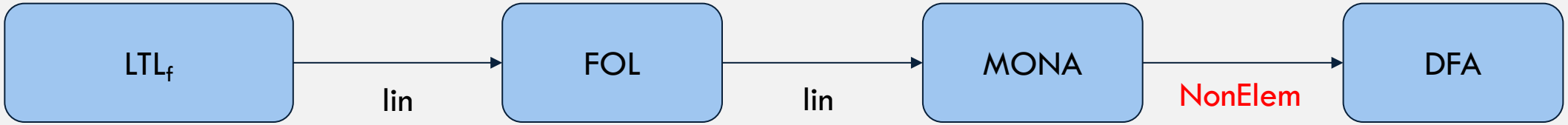
(online software for LTLf2DFA: `http://ltlf2dfa.diag.uniroma1.it`)
(online software for LDLf2DFA: `http://lydia.whitemech.it`)

LTLf to DFA (Techniques)

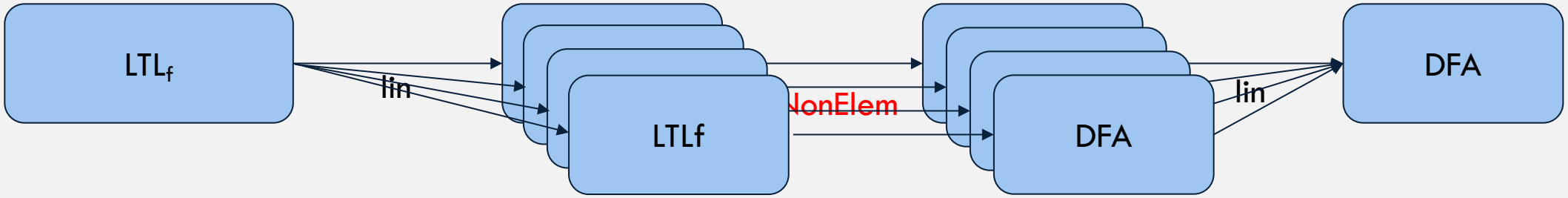
Monolitic tight bounds: [DeGiacomoVardi IJCAI2013/2015]



Monolitic via MONA [Zhu et al. IJCAI 2017]



Compositional [Bansal et al. AAAI2020], [DeGiacomoFavorito ICAPS2021]

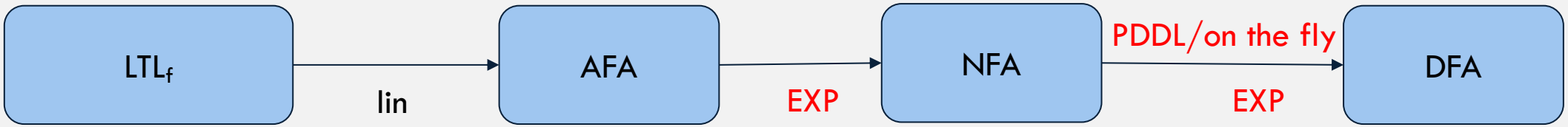


Better in practice!

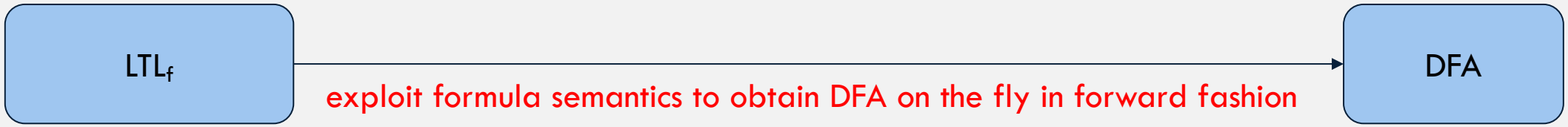
Online tool available! Monolitic via MONA < http://ltlf2dfa.diag.uniroma1.it>; Compositional <http://lydia.whitemech.it>

LTLf to DFA (Techniques)

Use planning for doing deteminization on the fly [Camacho et al ICAPS 2018]



On the fly forward fashion [Xiao et al. AAAI2021], [DeGiacomo et al. 2022]



Based on "next normal form" or "progression" [BacchusKabanzaAAAI1998]:

eventually Red iff Red or next eventually Red

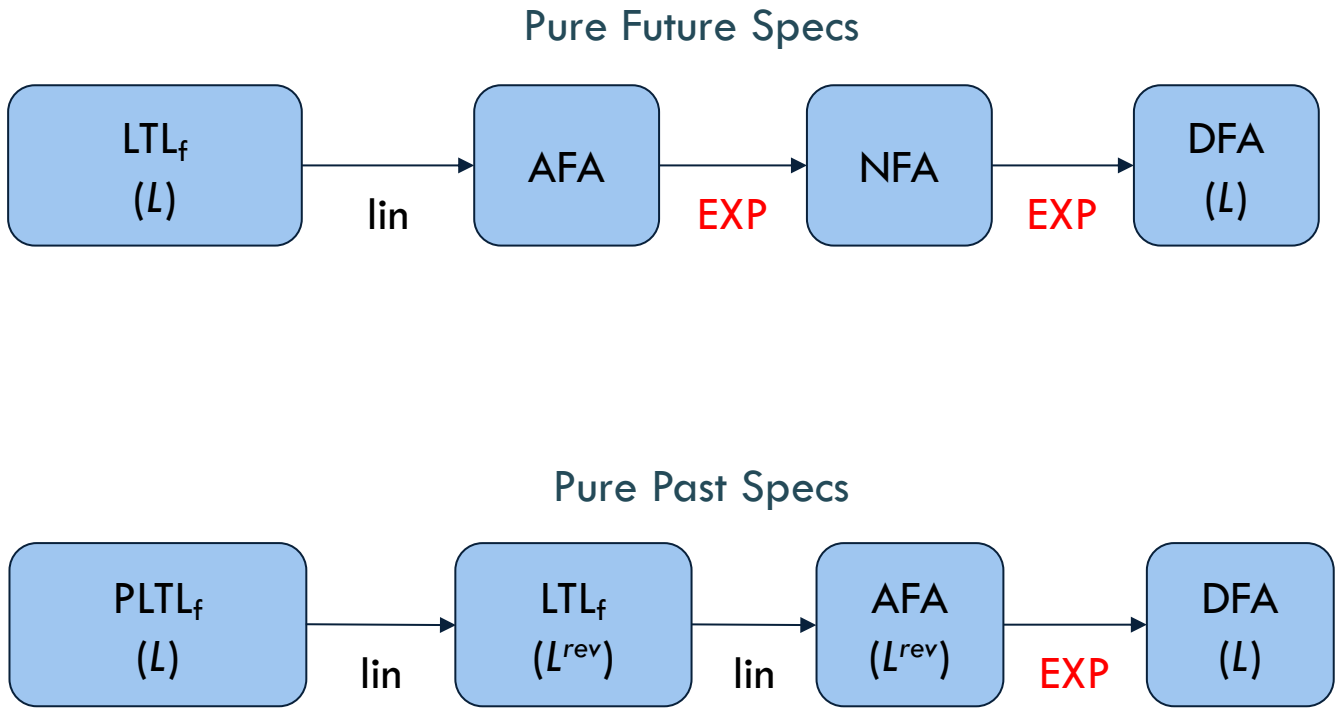
Important: transition must be "symbolic" i.e., propositional formulas

Note: LTLf cannot be polynomially translated to PDDL! (since 2EXP instead of 1EXP)

Pure Past LTLf

Pure past temporal specifications on finite traces

- Sometimes specifications are *easier* and *more natural* to express referring to the past
  ["The Glory of the Past" LichtensteinPnueliZuck1985]
  - Non-Markovian models [Gabaldon 2011]
  - Non-Markovian rewards in MDPs [Bacchus et al. 1996]
  - Normative properties in multi-agent systems [FisherWooldridge2005], [Knobbout et al. 2016], [Alechina et al. 2018]
- This is very convenient because we do have an exponential computational advantages in this cases



Given an AFA of k states for language L, there exists a DFA of at most 2^k states for language $L^{reverse}$ [Chandra et al. 1981]

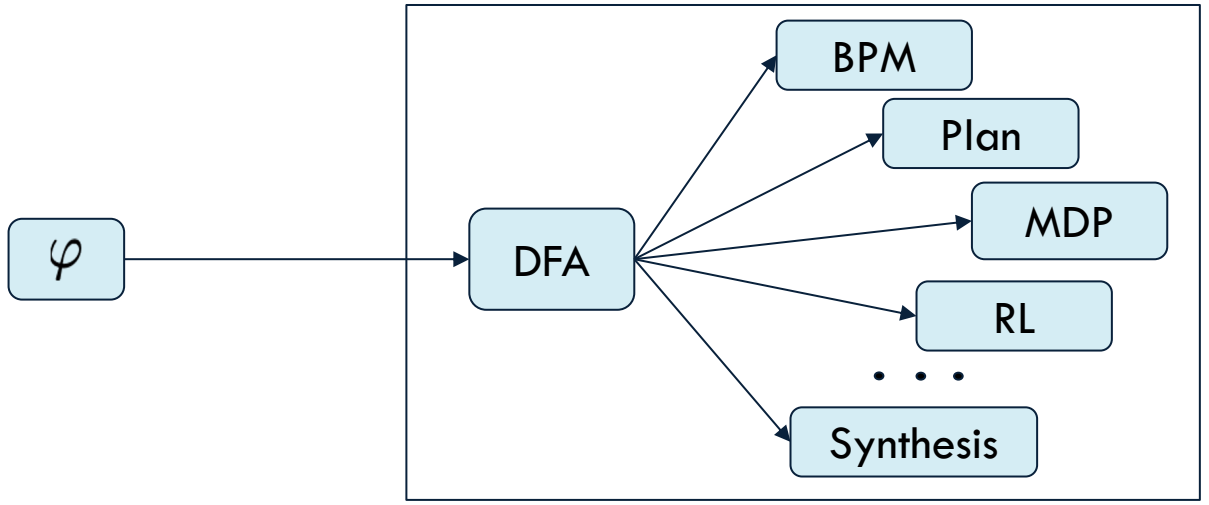
To undestand the practical impact see ICAPS2023

G. De Giacomo, A. Di Stasio, F. Fuggitti, and S. Rubin.
Pure-Past Linear Temporal and Dynamic Logic on Finite Traces. IJCAI 2020 Survey Track.

Several Applications of LTLf Specs

Many Applications:

- Declarative Process Specification in BPM
- Planning for temporally extended goals
- MDP with non-Markovian rewards
- Reinforcement Learning for non-Markovian tasks
- Several forms of Synthesis



Foundations of Framed Autonomy in AI-Augmented BPM Systems

SYNTHESIS

Program Synthesis

Program Synthesis

- Basic Idea: "Mechanical translation of human-understandable task specifications to a program that is known to meet the specifications." [Vardi - The Siren Song of Temporal Synthesis 2018]
- Classical vs. Reactive Synthesis:
  - Classical: Synthesize transformational programs [Green1969], [WaldingerLee1969], [Manna and Waldinger1980]
  - Reactive: Synthesize programs for interactive/reactive ongoing computations (protocols, operating systems, controllers, robots, etc.) [Church1963], [HarelPnueli1985], [AbadiLamportWolper1989], [PnueliRosner1989]

Reactive Synthesis

- Reactive synthesis is by now equipped with a elegant and comprehensive theory [EhlersLafortuneTripakisVardi2017], [Finkbeiner2018]
- Reactive synthesis is conceptually related to planning in fully observable nondeterministic domains (FOND) [DeGiacomoVardi2015], [DeGiacomoVardi2016], [DeGiacomoRubin2018], [CamachoTriantafillouMuiseBaierMcIlraith2017], [CamachoMuiseBaierMcIlraith2018], [CamachoBienvenuMcIlraith2019]

Planning and Reactive Synthesis

Planning in Fully Observable Nondeterministic domain

- fluents F (propositions) – controlled by the environment
- actions A (actions) – controlled by the agent
- domain D – specification of the dynamics
- goal G – propositional formula on fluents describing desired state of affairs to be reached

Planning = game between two players

- arena: the domain
- players: the agent and the environment
- game: **agent** tries to force eventually reaching G no matter how other **environment** behave
- Plan = agent-strategy $(2^F)^* \rightarrow A$ to win the game

Algorithms

EXPTIME-complete.
But we have very good algorithms.
(The entire ICAPS community involved!)

Reactive Synthesis

- inputs X (propositions) – controlled by the environment
- outputs Y (propositions) – controlled by the agent
- domain – not considered
- goal φ – arbitrary LTL (or other temporal logic specification) on both X and Y

Synthesis = game between two players

- arena: unconstraint! clique among all possible assignments for X and Y
- players: the agent and the environment
- game: **agent** tries to force a play that satisfies φ no matter how other **environment** behave.
- Winning strategy = agent-strategy $(2^X)^* \rightarrow 2^Y$ to win the game.

Algorithms

2EXPTIME-complete.
But we only have non-scalable algorithms.
(In spite of 30 years of research!)

Solving Reactive Synthesis on Infinite Traces

Synthesis for general linear time logic (LTL) specifications does not scale.

Solving reactive synthesis

Algorithm for LTL synthesis

Given LTL formula φ
1: Compute corresponding Buchi Nondeterministic Aut. (NBW) (exponential)
2: Determinize NBW into Deterministic parity Aut. (DPW) (exp in states, poly in priorities)
3: Synthesize winning strategy for parity game (poly in states, exp in priorities)
Return strategy

Reactive synthesis is 2EXPTIME-complete, but more importantly the problems are:

- The determinization in Step 2: no scalable algorithm exists for it yet.
  - From 9-state NBW to 1,059,057-state DRW [AlthoffThomasWallmeier2005]
  - No symbolic algorithms
- Solving parity games requires computing nested fixpoints (possibly exp many)

Solving Reactive Synthesis on Finite Traces

It's a game agent vs env!
- *Build the arena*
- *Play to win!*

Reactive synthesis

- **Framework:** We partition the set $\mathcal{P}$ of propositions into two disjoint sets:
  - $\mathcal{X}$ controlled by environment
  - $\mathcal{Y}$ controlled by agent

  Can the agent set the values of $\mathcal{Y}$ in such a way that for all possible values of $\mathcal{X}$ a certain $\text{LTL}_f/\text{LDL}_f$ formula remains true?
- **Solution:** compute a function $f : (2^{\mathcal{X}})^* \to 2^{\mathcal{Y}}$ such that for all generated traces π with X_i arbitrary and $Y_i = f(\pi_{\mathcal{X}}|_i)$, we have that π satisfies the formula ϕ.

Algorithm for $\text{LDL}_f/\text{LTL}_f$ synthesis

1: Given $\text{LTL}_f/\text{LDL}_f$ formula φ
2: Compute AFW for φ (linear)
2: Compute corresponding NFA (exponential)
3: Determinize NFA to DFA (exponential)
4: Synthesize winning strategy for DFA game (linear)
5: Return strategy

Thm: $\text{LTL}_f/\text{LDL}_f$ synthesis is 2-EXPTIME-complete.

Same as for infinite traces

Nondeterministic Domains as Deterministic Automata

Nondeterministic domain (including initial state)

$\mathcal{D} = (2^{\mathcal{F}}, \mathcal{A}, s_0, \delta, \alpha)$ where:

- $\mathcal{F}$ **fluents** (atomic propositions)
- $\mathcal{A}$ **actions** (atomic symbols)
- $2^{\mathcal{F}}$ set of states
- s_0 initial state (initial assignment to fluents)
- $\alpha(s) \subseteq \mathcal{A}$ represents **action preconditions**
- $\delta(s, a, s')$ with $a \in \alpha(s)$ represents **action effects (including frame)**.

Automaton A_D for $\mathcal{D}$ is a DFA!!!

$A_{\mathcal{D}} = (2^{\mathcal{F} \cup \mathcal{A}}, (2^{\mathcal{F}} \cup \{s_{init}\}), s_{init}, \varrho, F)$ where:

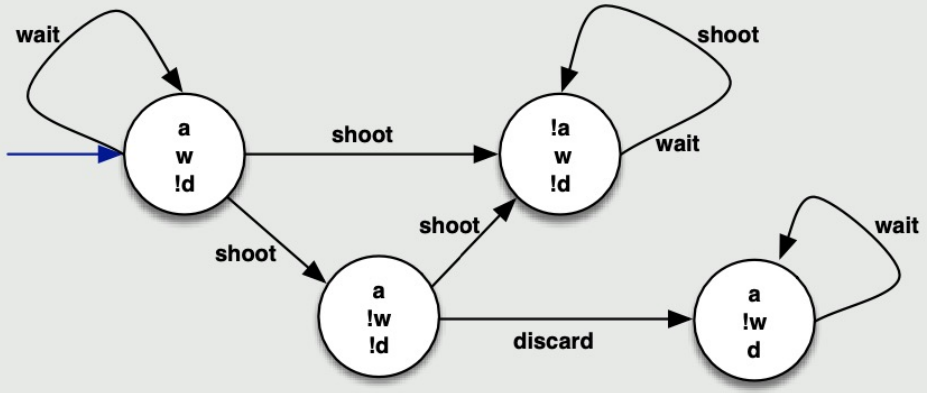
- $2^{\mathcal{F} \cup \mathcal{A}}$ alphabet (actions $\mathcal{A}$ include dummy $start$ action)
- $2^{\mathcal{F}} \cup \{s_{init}\}$ set of states
- s_{init} dummy initial state
- $F = 2^{\mathcal{F}}$ (all states of the domain are final)
- $\rho(s, [a, s']) = s'$ **with** $a \in \alpha(s)$**, and** $\delta(s, a, s')$ $\quad \rho(s_{init}, [start, s_0]) = s_0$

(notation: $[a, s']$ stands for $\{a\} \cup s'$)

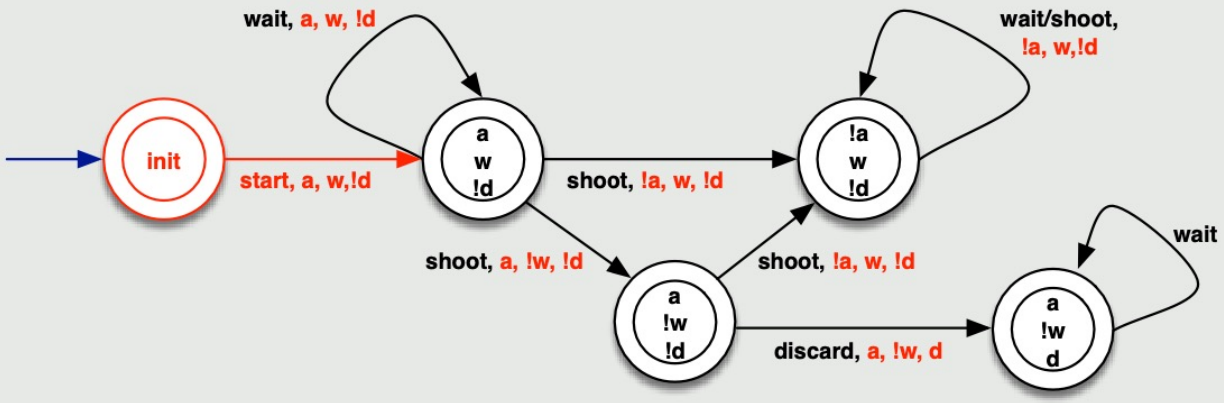
Nondeterministic Domains as Deterministic Automata

Example (Simplified Yale shooting domain)

- Domain $\mathcal{D}$:



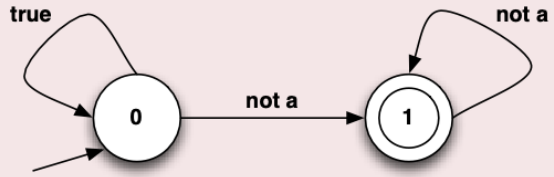
- DFA $A_{\mathcal{D}}$:



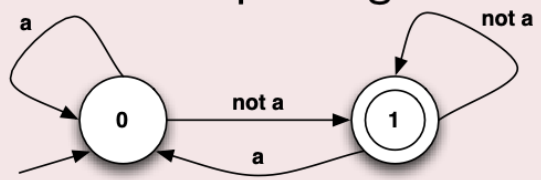
Planning in Nondeterministic Domains for Arbitrary LTLf Goals

In general, we need first to determinize the NFA for $\mathrm{LTL}_f/\mathrm{LDL}_f$ formula

NFA for $\Diamond\Box\neg a$



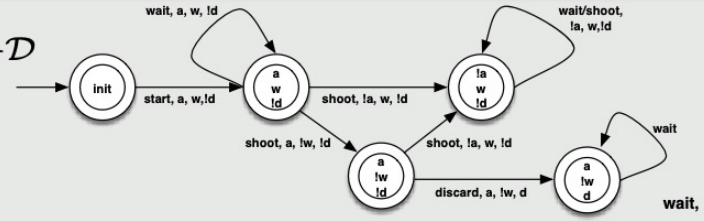
corresponding DFA



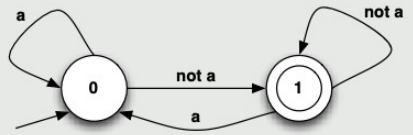
(DFA can be exponential in NFA in general)

Example (Simplified Yale shooting domain)

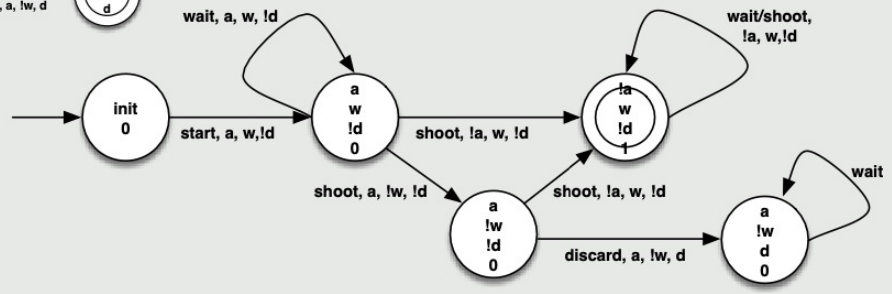
$A_{\mathcal{D}}$



$A_{\Diamond\Box\neg a}$



$A_{\mathcal{D}} \times A_{\Diamond\Box\neg a}$:



strategy

$init, 0$	$\rightarrow$	$start$
$a, w, \neg d, 0$	$\rightarrow$	$shoot$
$a, \neg w, \neg d, 0$	$\rightarrow$	$shoot$
$\neg a, w, \neg d, 1$	$\rightarrow$	win!

Planning in Nondeterministic Domains for LTLf Goals

DFA games

A DFA **game** $\mathcal{G} = (2^{\mathcal{F} \cup \mathcal{A}}, S, s_{init}, \varrho, F)$, is such that:

- $\mathcal{F}$ controlled by environment; $\mathcal{A}$ controlled by agent;
- $2^{\mathcal{F} \cup \mathcal{A}}$, alphabet of game;
- S, states of game;
- s_{init}, initial state of game;
- $\varrho : S \times 2^{\mathcal{F} \cup \mathcal{A}} \rightarrow S$, transition function of the game: given current state s and a choice of action a and resulting fluents values E the resulting state of game is $\varrho(s, [a, E]) = s'$;
- F, final states of game, where game can be considered terminated.

*This is the **game area**, in which agent and env will play!*

*It **is not only the domain**, but it is obtained from the **domain** and the **DFA of the formula***

Winning Strategy:

- A play is **winning** for the agent if such a play leads from the initial to a final state.
- A **strategy** for the agent is a function $f : (2^{\mathcal{F}})^* \rightarrow \mathcal{A}$ that, given a **history of choices from the environment**, decides which action $\mathcal{A}$ to do next.
- A **winning strategy** is a strategy $f : (2^{\mathcal{F}})^* \rightarrow \mathcal{A}$ such that for all traces π with $a_i = f(\pi_{\mathcal{F}}|_i)$ we have that π leads to a final state of $\mathcal{G}$.

Planning in Nondeterministic Domains for LTLf Goals

Winning states for DFA games

Let denote the **set of final states** of $\mathcal{G}$ as:

$$[\![F]\!] = \{s \in \mathcal{S} \mid s \models F\}$$

and let's define the **(adversarial preimage of a set $\mathcal{E}$** the following function:

$$PreAdv(\mathcal{E}) = \{s \in \mathcal{S} \mid \exists a \in \alpha(s).\forall s' \in \mathcal{S}.\, \delta(s,a,s') \supset s' \in \mathcal{E}\}$$

Compute the set Win of winning states of DFA game, i.e., states from which the agent can reach the final states F, by **least-fixpoint**:

- $Win_0 = [\![F]\!]$ (the final states)
- $Win_{i+1} = Win_i \cup PreAdv(Win_i)$
- $Win = \bigcup_i Win_i$

$W_{old} := \emptyset$
$W := [\![F]\!]$
while $(W \neq W_{old})\{$
  $W_{old} := W$
  $W := W \cup PreAdv(W)$
$\}$
return W

(Computing Win is linear in the number of states in $\mathcal{G}$)

Computing the winning strategy

Let's define $\omega : S \to 2^{\mathcal{A}}$ as: $\omega(s) = \{a \in \alpha(s) \mid \text{ if } s \in Win_{i+1} - Win_i \text{ then } \forall s'.\delta(s,a,s') \supset s' \in Win_i\}$

- **Every way** of restricting $\omega(s)$ to return only one action (chosen arbitrarily) gives a **winning strategy** for $\mathcal{G}$.
- Note s **is a state of the game**! not of the domain only!
  To phrase ω wrt the domain only, we need to return a **stateful transducer** with transitions from the game.

Planning in Nondeterministic Domains for LTLf Goals

FOND for LTL_f goals

Algorithm: FOND for $\text{LTL}_f/\text{LDL}_f$ goals

1: Given a FOND domain $\mathcal{D}$ and an $\text{LTL}_f/\text{LDL}_f$ goal φ
2: Compute DFA A_φ for φ (double exponential)
3: Compute product of $\mathcal{D}$ and A_φ (polynomial)
4: Synthesize winning strategy for DFA game (linear)
5: Return strategy

It's a game agent vs env!
- *Build the arena*
- *Play to win!*

Theorem ([DeGiacomoRubinIJCAI18])

FOND for $\text{LTL}_f/\text{LDL}_f$ goals is:

- *EXPTIME-complete in the domain (assuming a logarithmic representation as in PDDL);*
- *2EXPTIME-complete in the goal.*

Same as classic FOND

Note we have separated costs in the model (DOM) and the task GOAL!
(c.f. data vs query complexity in Databases)

Deliberate how to act in a Partially Controllable Environment: FOND Planning/Synthesis for LTLf Goals

FOND for LTL_f goals

It's a game agent vs env!
- *Build the arena*
- *Play to win!*

Algorithm: FOND for $\text{LTL}_f/\text{LDL}_f$ goals

1: Given a FOND domain $\mathcal{D}$ and an $\text{LTL}_f/\text{LDL}_f$ goal φ
2: Compute DFA A_φ for φ (double exponential)
3: Compute product of $\mathcal{D}$ and A_φ (polynomial)
4: Synthesize winning strategy for DFA game (linear)
5: Return strategy

Theorem ([DeGiacomoRubinIJCAI18])

FOND for $\text{LTL}_f/\text{LDL}_f$ goals is:

- *EXPTIME-complete in the domain (assuming a logarithmic representation as in PDDL);*
- *2EXPTIME-complete in the goal.*

Same as classic FOND

Note we have **separated cost** in the **model** (the doman) from that in the **task** (the goal)!
(cf. data vs query complexity [ChandraHarel1980], [Vardi1982], [AbiteboulHullVianu1995])

Deliberate how to act in a Partially Controllable Environment: Reactive Synthesis in Formal Methods

It's a game agent vs env!
- *Build the arena*
- *Play to win!*

Reactive synthesis

- **Framework:** We partition the set $\mathcal{P}$ of propositions into two disjoint sets:
  - $\mathcal{X}$ controlled by environment
  - $\mathcal{Y}$ controlled by agent

  Can the agent set the values of $\mathcal{Y}$ in such a way that for all possible values of $\mathcal{X}$ a certain $\text{LTL}_f/\text{LDL}_f$ formula remains true?
- **Solution:** compute a function $f : (2^{\mathcal{X}})^* \to 2^{\mathcal{Y}}$ such that for all generated traces π with X_i arbitrary and $Y_i = f(\pi_{\mathcal{X}}|_i)$, we have that π satisfies the formula ϕ.

Algorithm for $\text{LDL}_f/\text{LTL}_f$ synthesis

1: Given $\text{LTL}_f/\text{LDL}_f$ formula φ
2: Compute AFW for φ (linear)
2: Compute corresponding NFA (exponential)
3: Determinize NFA to DFA (exponential)
4: Synthesize winning strategy for DFA game (linear)
5: Return strategy

Thm: $\text{LTL}_f/\text{LDL}_f$ synthesis is 2-EXPTIME-complete.

Same as for infinite traces

Deliberate how to act in a Partially Controllable Environment: FOND Planning/Synthesis for LTLf Goals

FOND for LTL_f goals

It's a game agent vs env!
- *Build the arena*
- *Play to win!*

Algorithm: FOND for $\text{LTL}_f/\text{LDL}_f$ goals

1: Given a FOND domain $\mathcal{D}$ and an $\text{LTL}_f/\text{LDL}_f$ goal φ
2: Compute DFA A_φ for φ (double exponential)
3: Compute product of $\mathcal{D}$ and A_φ (polynomial)
4: Synthesize winning strategy for DFA game (linear)
5: Return strategy

Theorem ([DeGiacomoRubinIJCAI18])

FOND for $\text{LTL}_f/\text{LDL}_f$ goals is:
- *EXPTIME-complete in the domain (assuming a logarithmic representation as in PDDL);*
- *2EXPTIME-complete in the goal.*

Note we have **separated cost** in the **model** (the doman) from that in the **task** (the goal)!
(cf. data vs query complexity [ChandraHarel1980], [Vardi1982], [AbiteboulHullVianu1995])

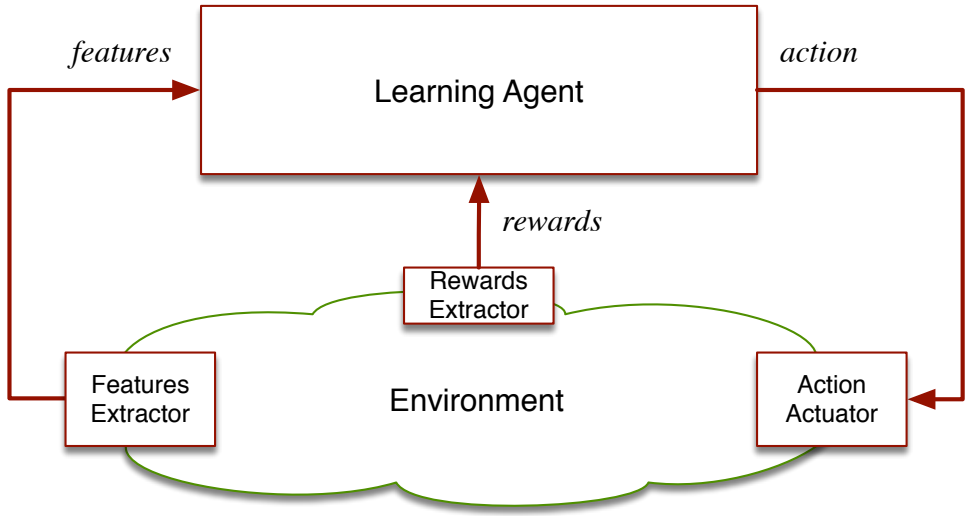
Foundations of Framed Autonomy in AI-Augmented BPM Systems

MERGING REASONING AND LEARNING

Learning and Reasoning

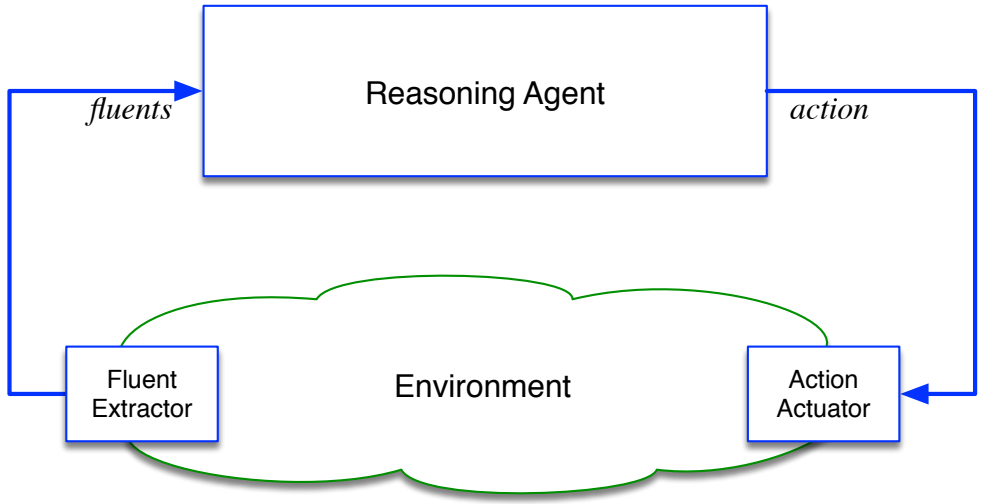
Learning agent:

- Senses and acts on the environment
- Gets rewards when right
- Does reinforcement learning



Reasoning agent:

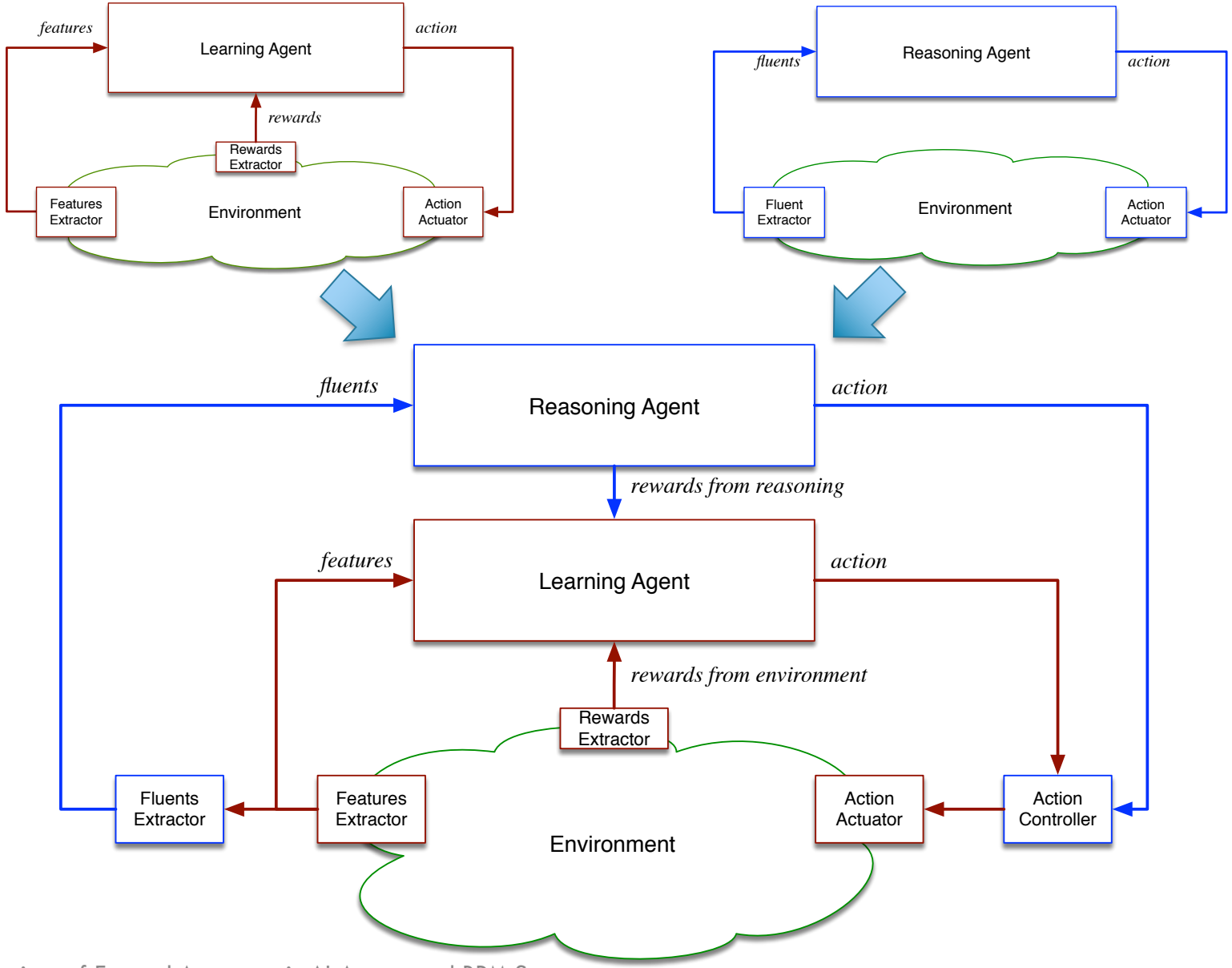
- Senses and acts on the environment
- Has models of its environment and tasks
- Does reasoning and planning



Merging Learning and Reasoning

Merging:

- Learning agent
  - Does reinforcement learning
  - Possibly deep reinforcement learning
- Reasoning agent
  - Does reasoning
  - Possibly on temporal specification as in formal methods



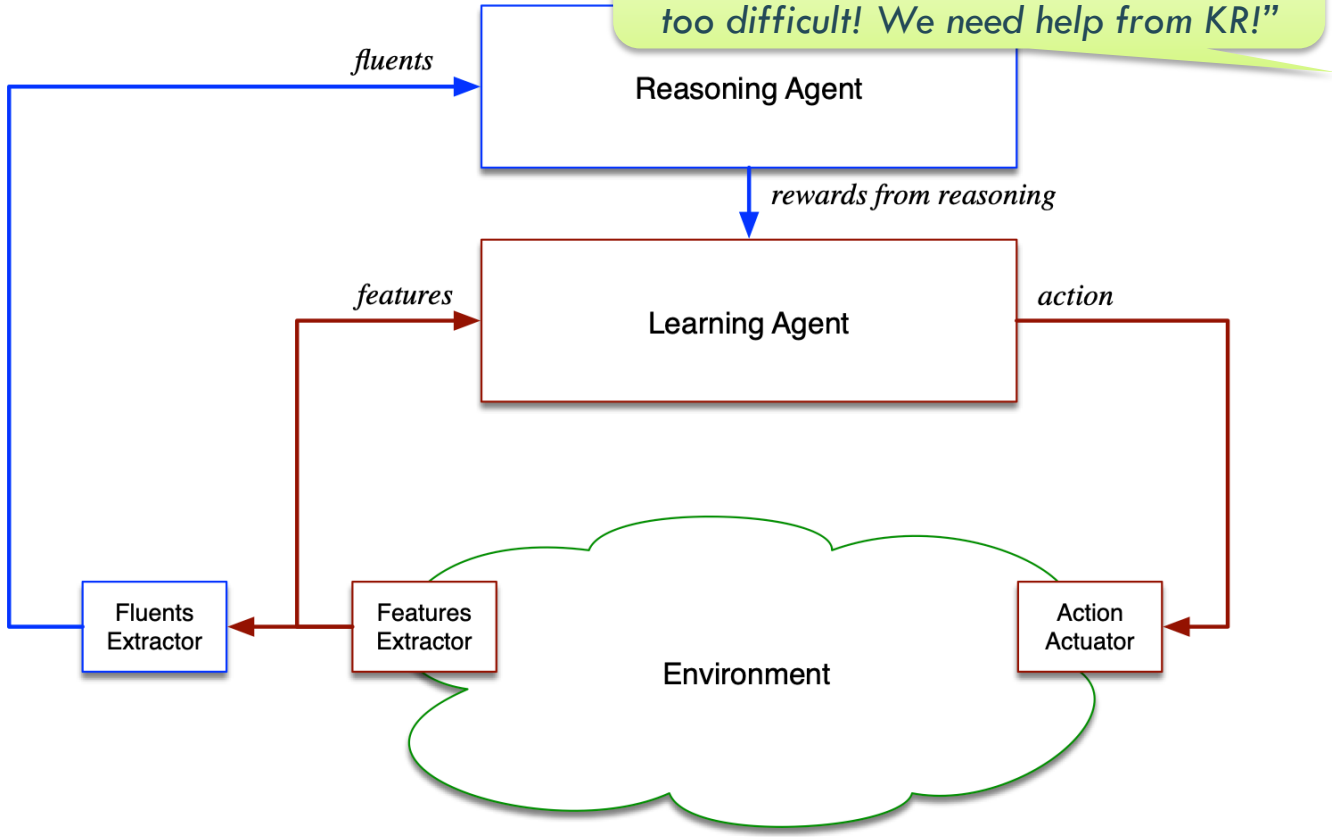
Reinforcement Learning with LTLf non-Markovian Rewards

MDPs with non-Markovian rewards

- **Learning agent:** $\mathcal{M} = (S_{ag}, A_{ag}, Tr_{ag}, \cancel{R_{ag}})$
  MDP without rewards
- **Reasoning agent:** $\mathcal{R} = (\mathcal{L}, \{(\varphi_i, r_i)\}_{i=1}^{m})$
  φ_i in LTLf/LDLf ⇒ $\overline{R}_{ag} : (S_{ag}, A_{ag})^* \rightarrow \mathbb{R}$
  non-Markovian rewards!
- Mapping between S_{ag} and $\mathcal{L}$

We can define equivalent MDP over an extended state space and do standard RL

Michael Littman @ IJCAI 2015:
"Coming up with rewards in MDPs is too difficult! We need help from KR!"



M. Littman. Programming agents via rewards. (Invited talk) IJCAI 2015.

R. Brafman, G. De Giacomo, F. Patrizi. LTLf /LDLf non-Markovian rewards. AAAI 2018.

A. Camacho, R. Icarte, T. Klassen, R. Valenzano, S. McIlraith. LTL and Beyond: Formal Languages for Reward Spec. in RL. IJCAI 2019.

Restraining Bolts

https://www.starwars.com/databank/restraining-bolt

RESTRAINING BOLT

A restraining bolt is a small cylindrical device that restricts a droid's actions when connected to its systems. Droid owners install restraining bolts to limit actions to a set of desired behaviors.

Two distinct representations of the environment

One for the agent
- by the designer of the agent

One for the restraining bolt
- by the authority imposing it

Restraining Bolts as Reasoning Agents

Double state representation (restraining bolts)

- Learning agent: $\mathcal{M} = (S_{ag}, A_{ag}, Tr_{ag}, \cancel{R_{ag}})$

  MDP without rewards

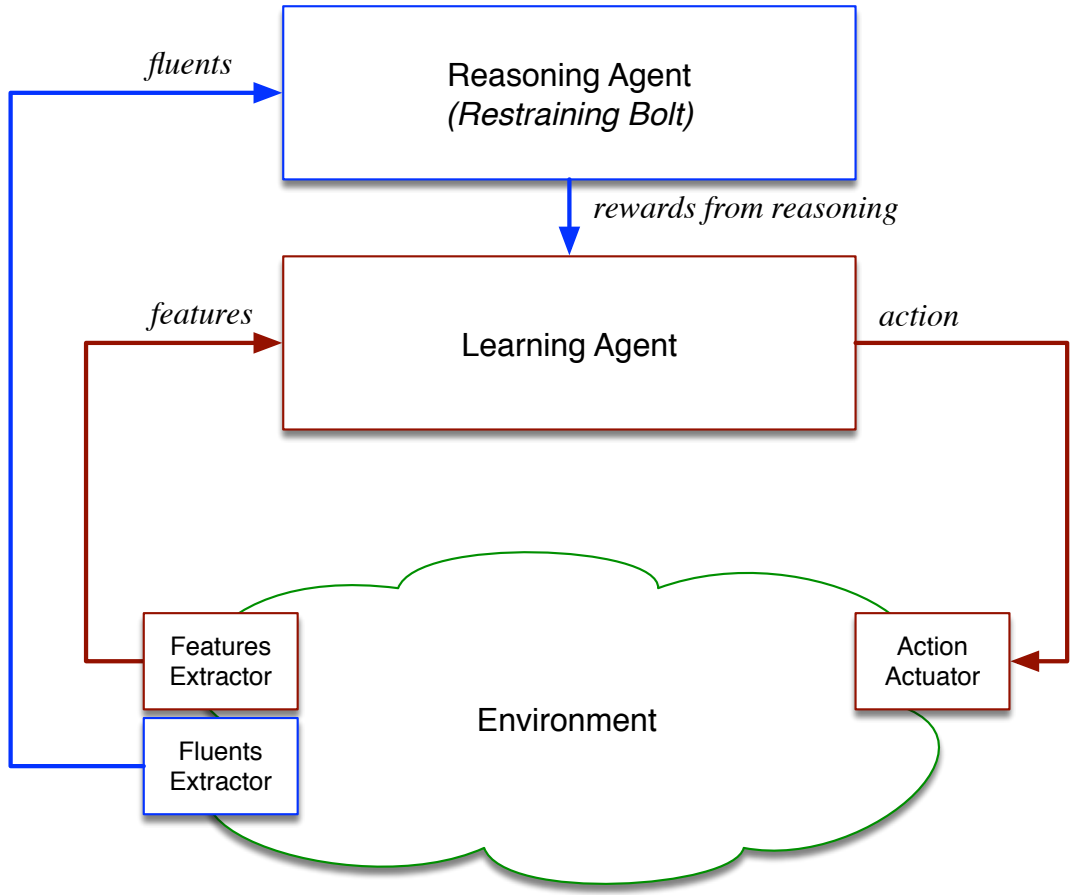
- Reasoning agent: $\mathcal{R} = (\mathcal{L}, \{(\varphi_i, r_i)\}_{i=1}^{m})$

  φ_i in LTLf/LDLf ⇒ $\overline{R}_{ag} : (S_{ag}, A_{ag})^* \rightarrow \mathbb{R}$

  non-Markovian rewards!

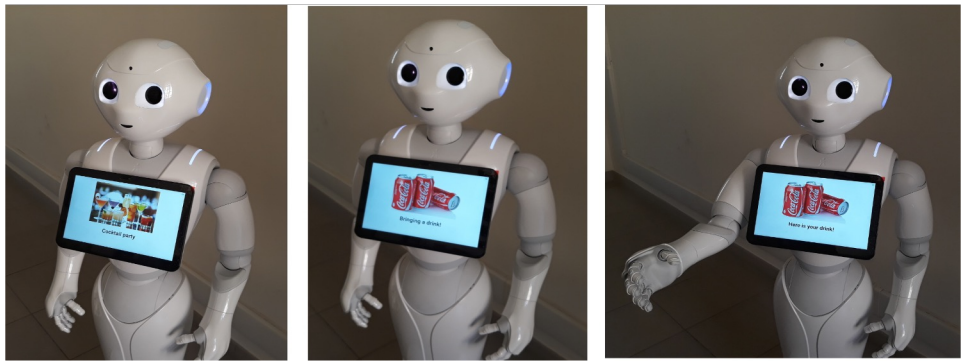
- ~~Mapping between S_{ag} and $\mathcal{L}$~~

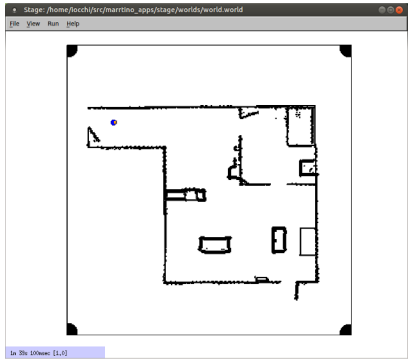
We can define equivalent MDP over an extended state space and do standard RL



G. De Giacomo, M. Favorito, L. Iocchi, and F. Patrizi. Foundations for Restraining Bolts: Reinforcement Learning with LTLf/LDLf Restraining Specifications. ICAPS 2019.

Example: Cocktail Party



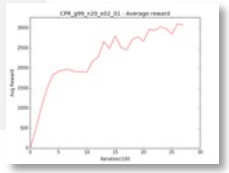


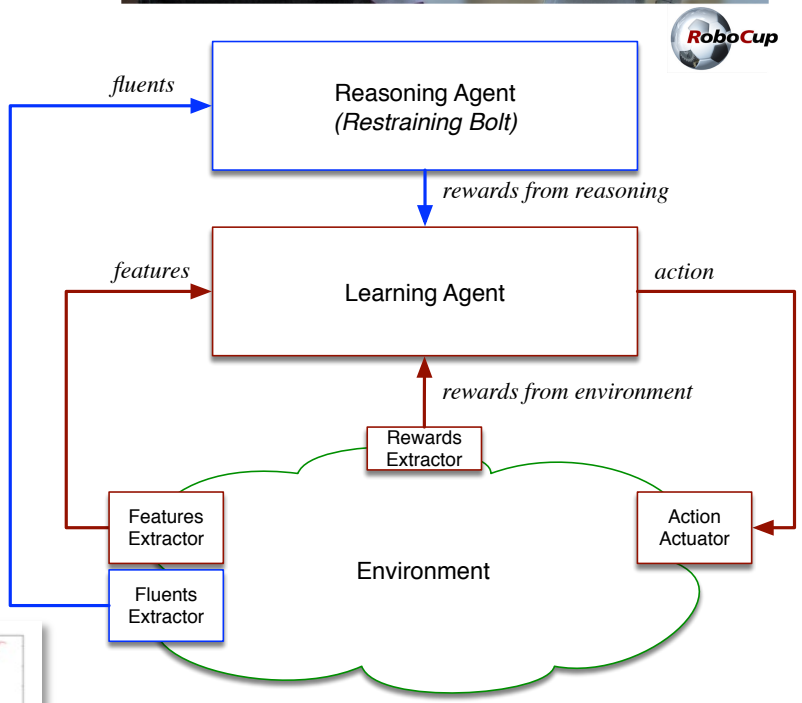
Learning Agent

- Features: robot's pose, location of objects (drinks and snacks), and location of people
- Actions: move in the environment, can grasp and deliver items to people
- Rewards: robot's navigation, deliver task is completed.

Restraining Bolt (Reasoning Agent)

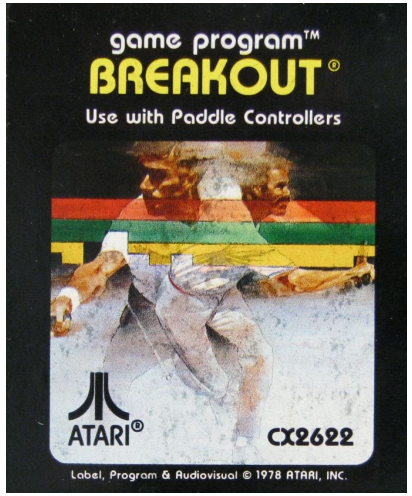
- Rewards: serve exactly one drink and one snack to every person, and do not serve alcoholic drinks to minors
- Fluents: identity and age of people, and received items *(uses Microsoft Cognitive Services Face API to provide information)*





https://sites.google.com/diag.uniroma1.it/restraining-bolt

Example: Breakout

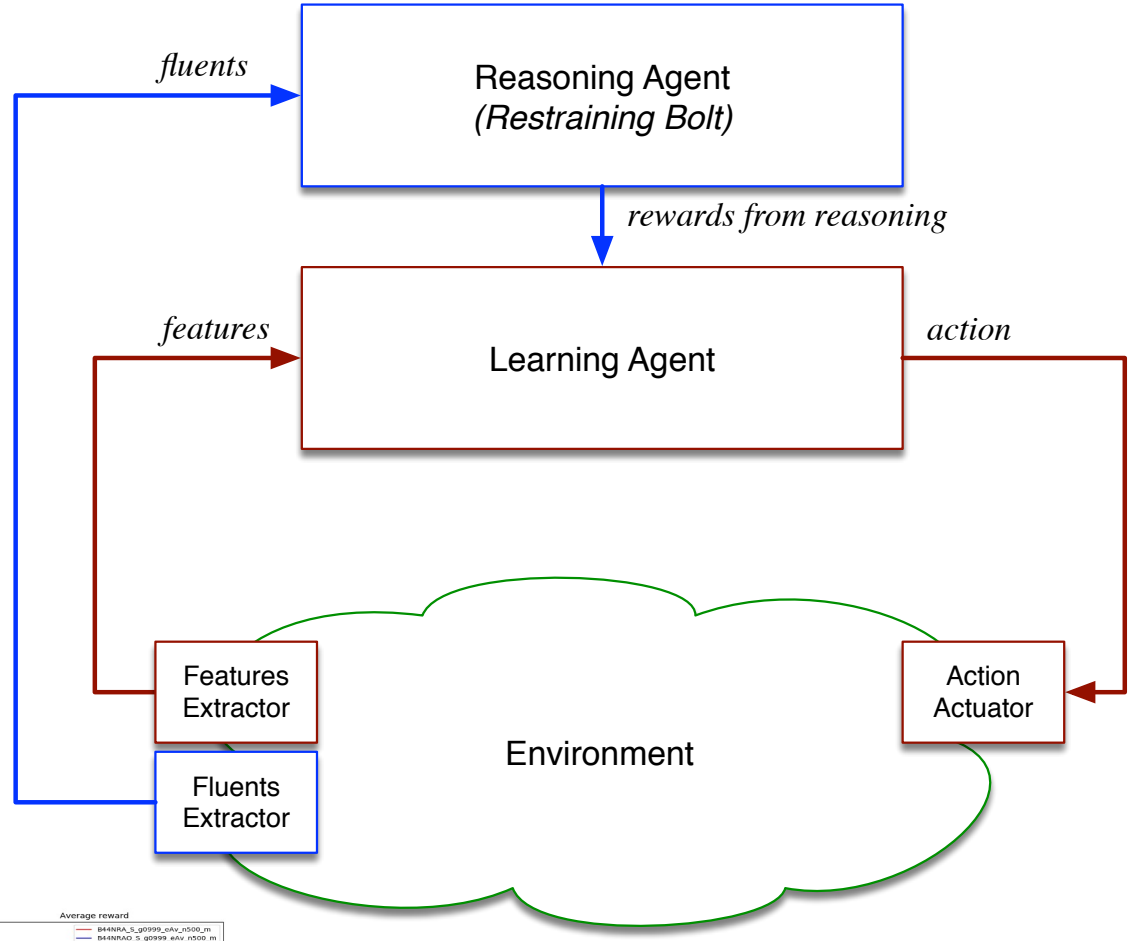


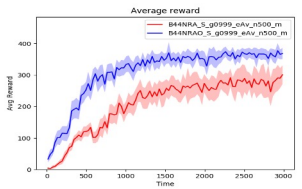
Learning Agent

- Features: paddle position, ball speed/position
- Actions: move the paddle
- ~~Rewards: reward when a brick is hit~~

Restraining Bolt (Reasoning Agent)

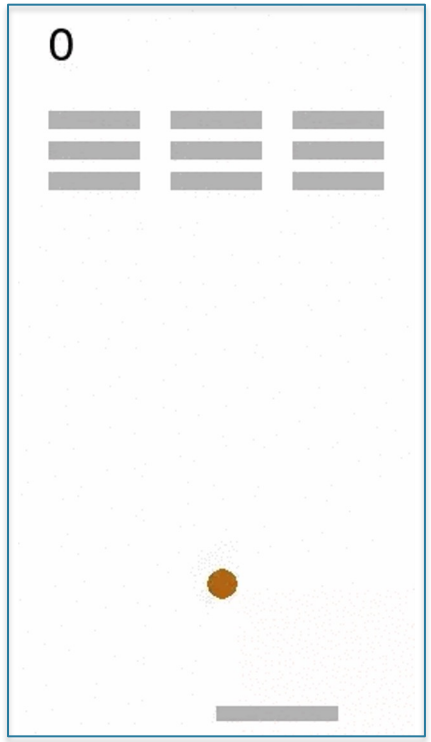
- Rewards: break one column at the time left to right (all bricks in column i must be removed before completing any other column j > i)
- Fluents: bricks/columns status (broken/not broken)



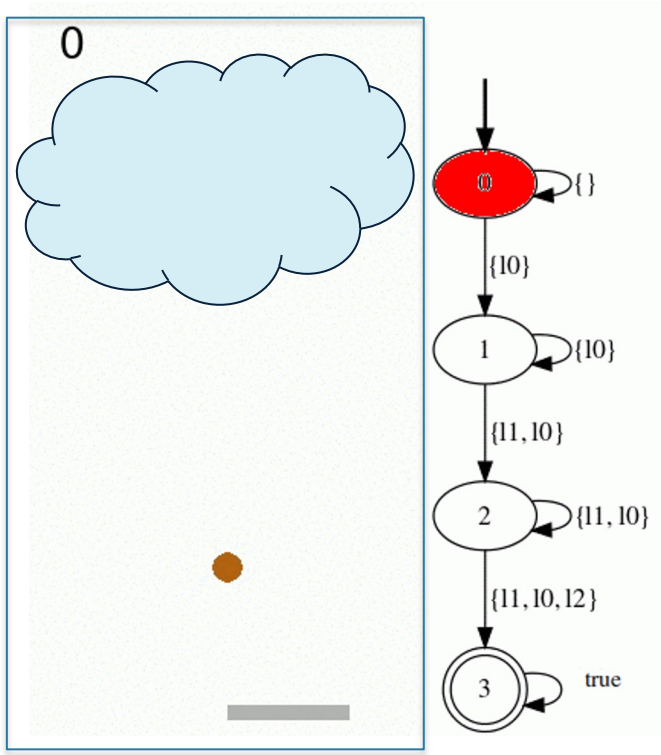


https://sites.google.com/diag.uniroma1.it/restraining-bolt

The Agent Ignores the Fluents!



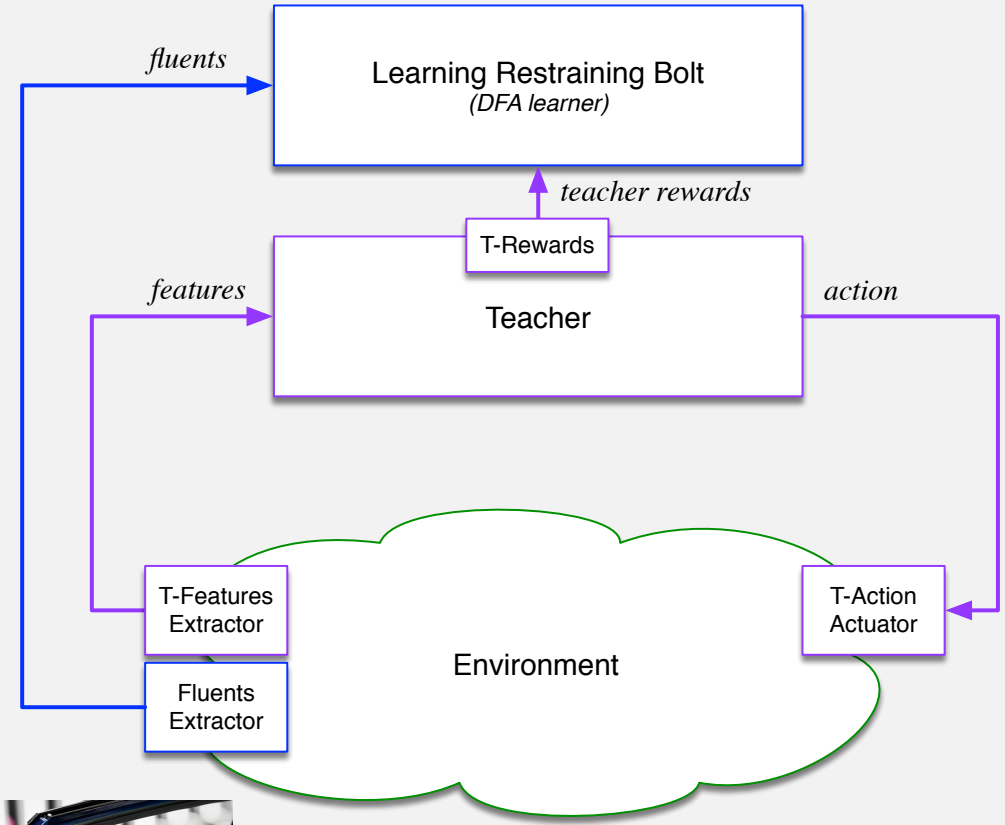
How the world is



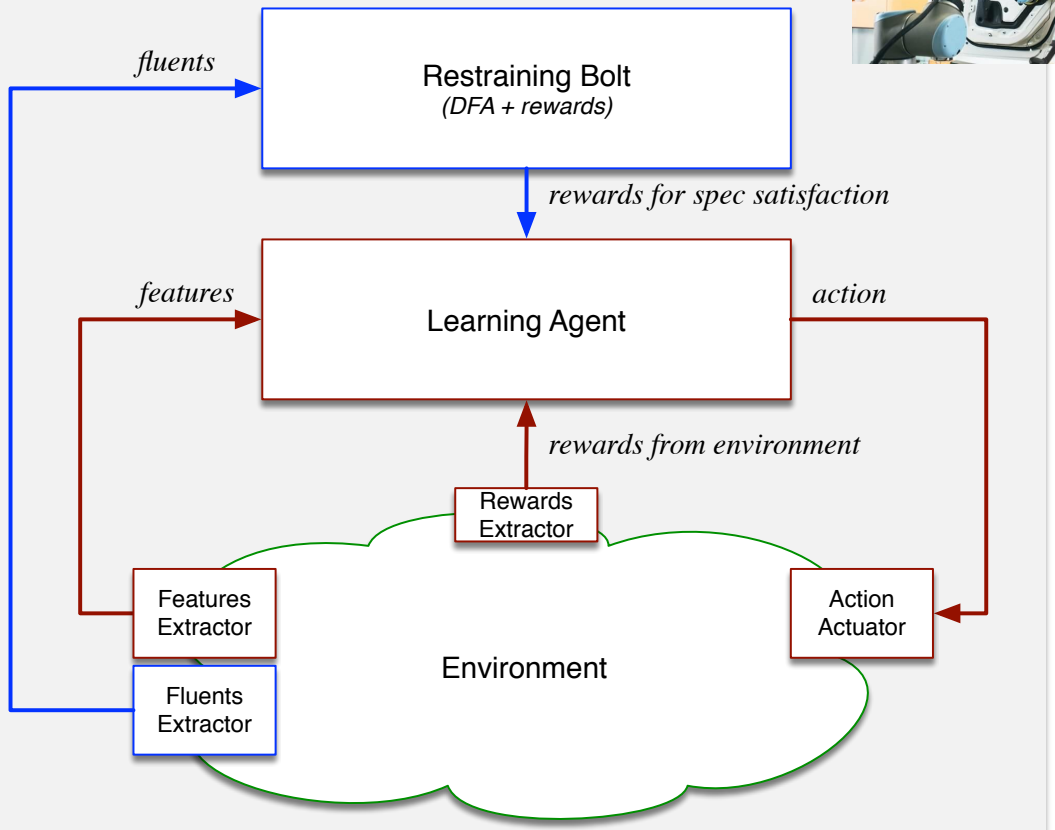
How the agent sees the world

Extensions: Imitation Learning

Learn a Restraining Bolt



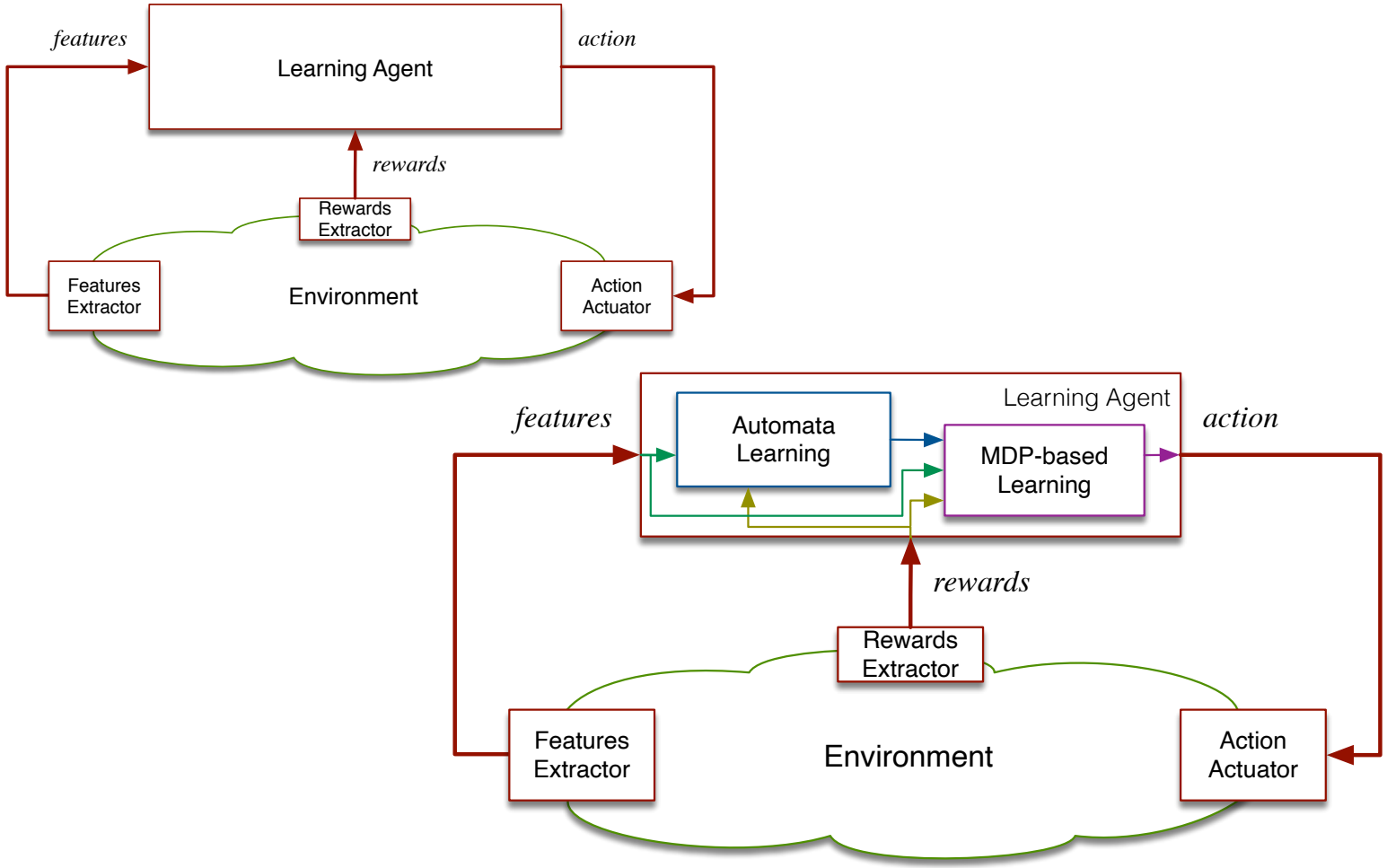
Use the Restraining Bolt



G. De Giacomo, M. Favorito, L. Iocchi, and F. Patrizi.
Imitation Learning over Heterogeneous Agents with Restraining Bolts. ICAPS 2020.
https://whitemech.github.io/Imitation-Learning-over-Heterogeneous-Agents-with-Restraining-Bolts

Challenge: Reinforcement Learning in non-Markovian Domains

- Reinforcement Learning is typically based on MDPs, i.e. on state-based domains
- Can we do handle non-Markovian dynamics (i.e., depending on the history) without postulating a priori existence of hidden variable, as in POMDPs?
- Use Regular Decision Processes (RDP) instead of MDPs
- Reinforcement Learning on RDPs requires simultaneously learning an automaton for the dynamics and an optimal policy wrt rewards:
  - Polynomial PAC-learnability
  - With no prior knowledge



R. Brafman, G. De Giacomo. Regular Decision Processes: A Model for Non-Markovian Domains. IJCAI 2019.

A. Ronca, G. De Giacomo. Efficient PAC Reinforcement Learning in Regular Decision Processes. IJCAI 2021

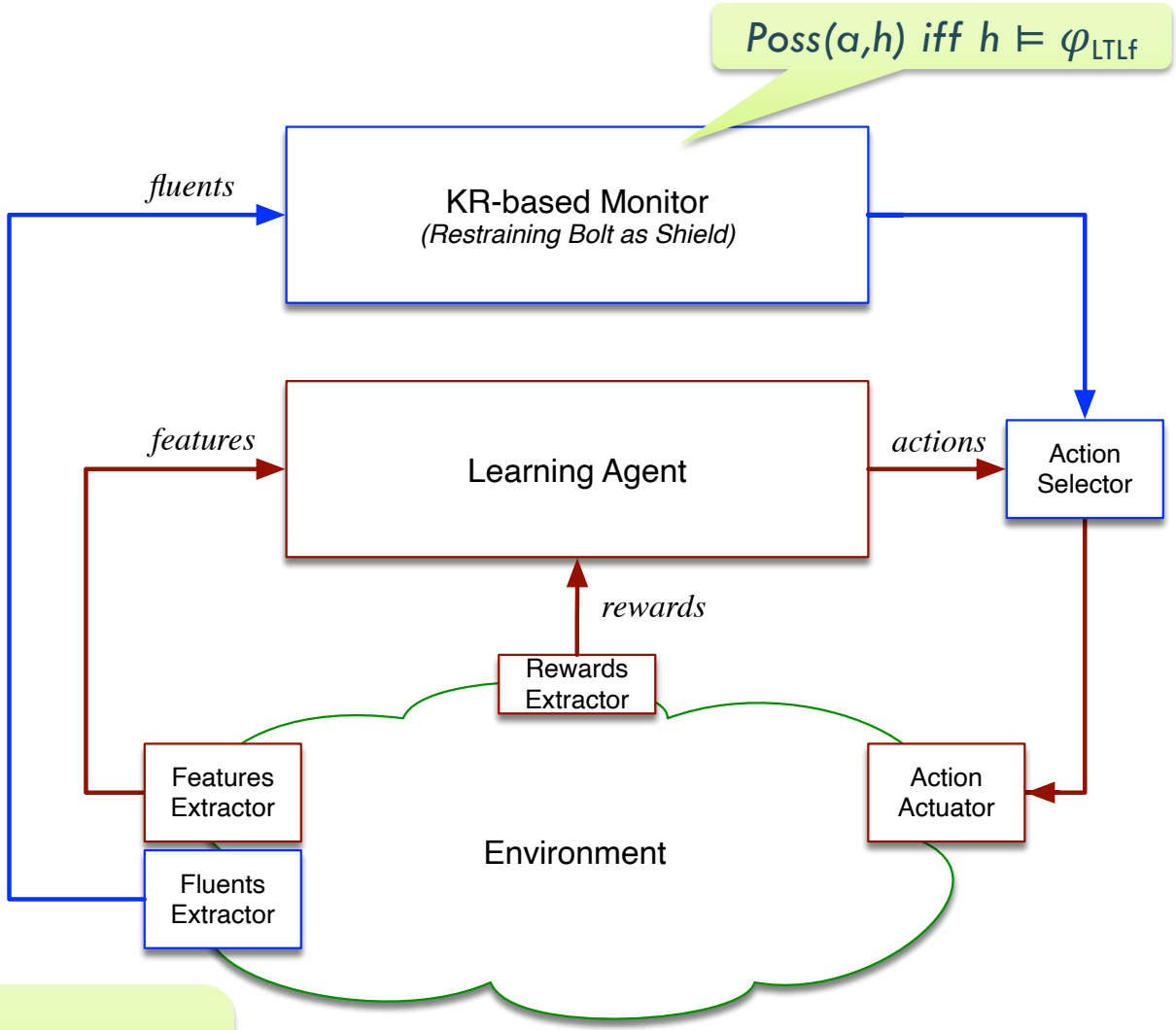
Restraining Bolts as Shields

Separate representations

- KR-based agent
- RL-based agent

Shielded execution

- KR-based agent acts as a monitor
- It disallows forbitten actions



Very related to "Framed Autonomy" in ABPMS, see
Augmented Business Process Management Systems: A Research Manifesto
https://arxiv.org/abs/2201.12855

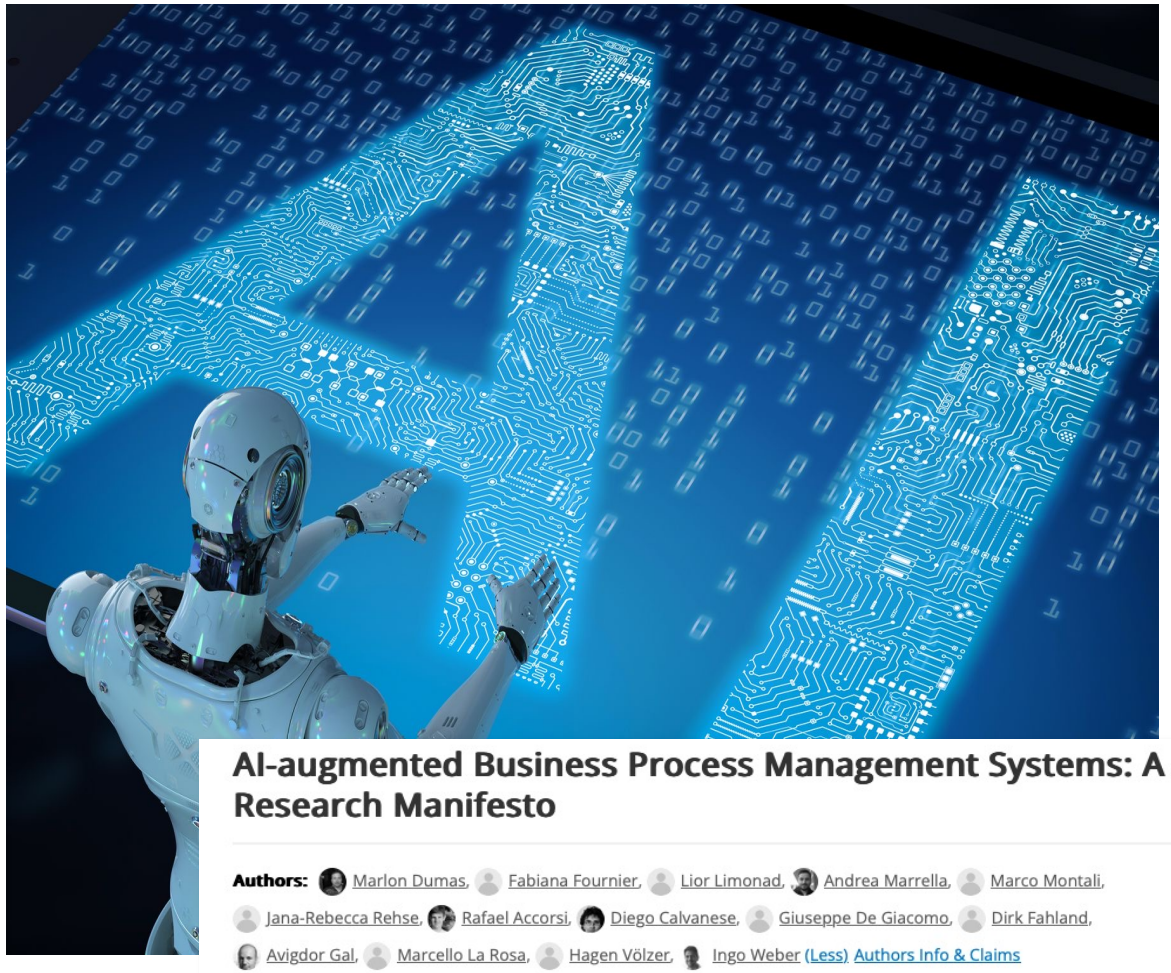
N. Alechina, G. De Giacomo, B. Logan, G. Varricchione.
*Thanks to TAILOR **Connectivity Fund**. In preparation.*

Foundations of Framed Autonomy in AI-Augmented BPM Systems

CONCLUSION

Conclusion

- Building Self-Deliberating Processes is one of the grand objectives of AI
- Important advancements from synergies among different areas of AI and CS:
  - Knowledge representation and reasoning
  - Planning
  - Multi-agent systems
  - Sequential decision making (MDPs)
  - Reinforcement learning
  - Formal methods
  - BPM
- Merging reasoning and learning is a new trend in AI: Novel results are available
- There are strong indications that Self-Deliberation may become a big thing in BPM!



AI-augmented Business Process Management Systems: A Research Manifesto

Authors: Marlon Dumas, Fabiana Fournier, Lior Limonad, Andrea Marrella, Marco Montali, Jana-Rebecca Rehse, Rafael Accorsi, Diego Calvanese, Giuseppe De Giacomo, Dirk Fahland, Avigdor Gal, Marcello La Rosa, Hagen Völzer, Ingo Weber (Less) Authors Info & Claims

ACM Transactions on Management Information Systems, Volume 14, Issue 1 • March 2023 • Article No.: 11, pp 1–19 • https://doi.org/10.1145/3576047