SAPIENZA
UNIVERSITÀ DI ROMA

Restraining Bolts for Reinforcement Learning Agents (MDPs with LTL_f Goals + Restraining Bolts)

Giuseppe De Giacomo

University of Rome "La Sapienza", Italy

Reasoning Agents 2020/2021

Restraining Bolts for Reinforcement Learning Agents

INTRODUCTION

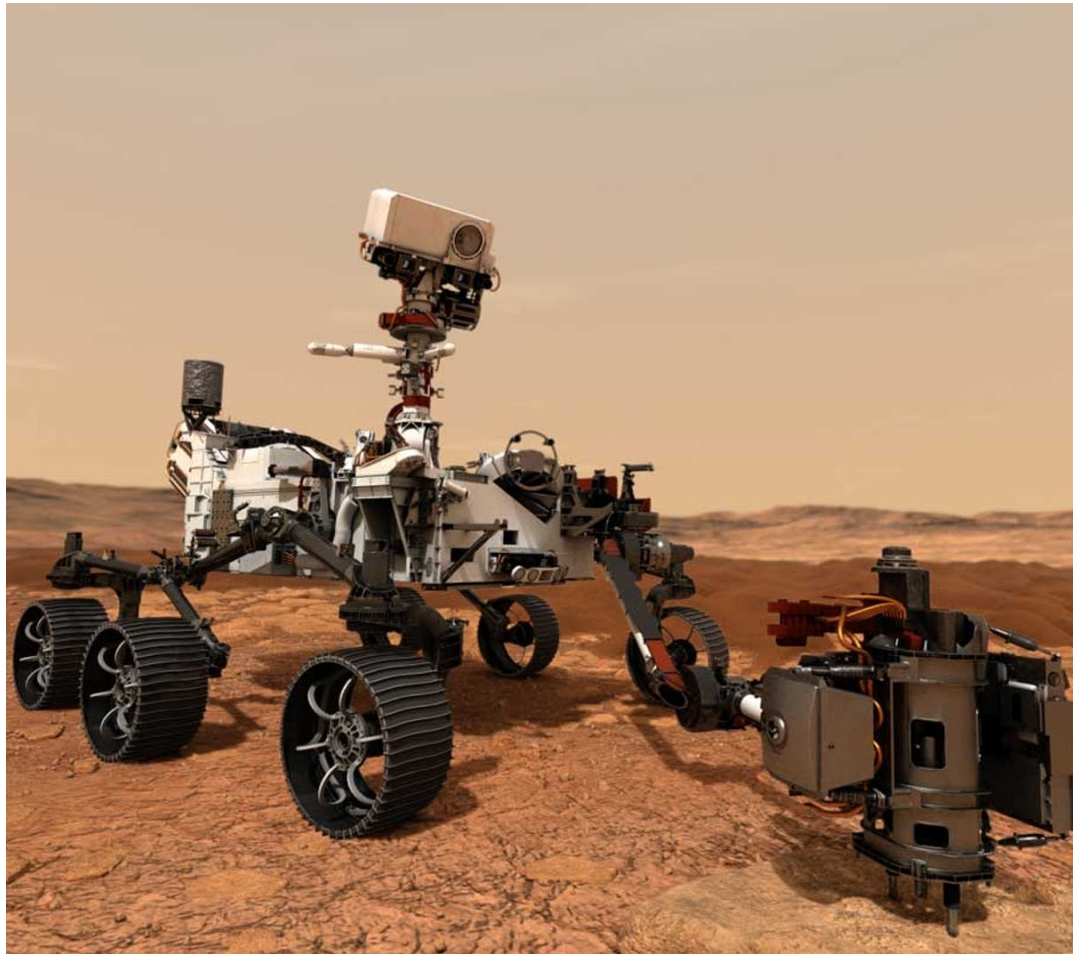
Autonomy in AI

- Autonomy is one of the grand objectives of AI.
- Aims at building autonomous agents/robots that operate in changing, incompletely known, unpredictable environments.
- Requires autonomous reasoning and planning capabilities, as well as learning from experience.

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.

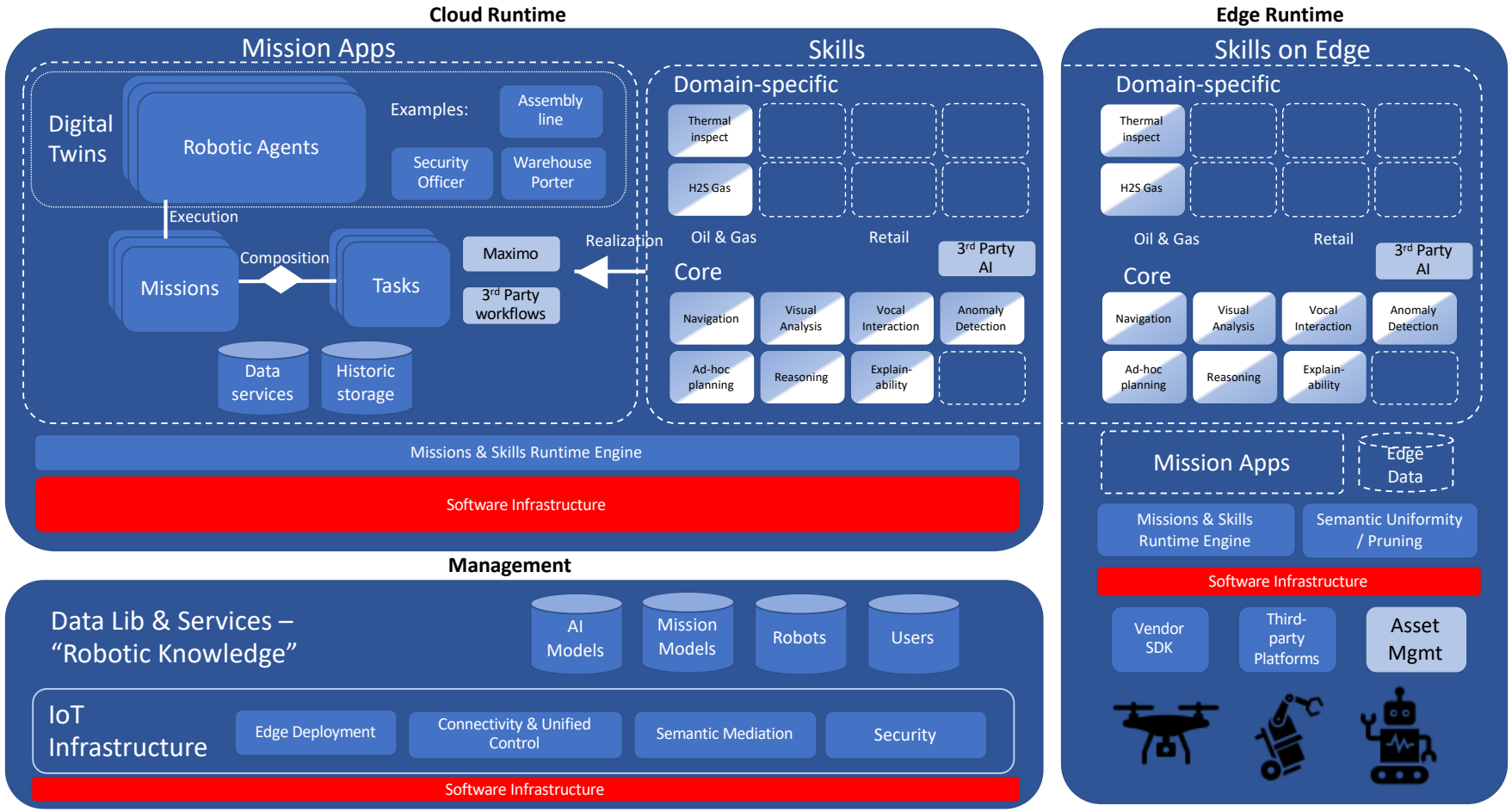


Autonomous Mobile Robots in Logistics

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

These robots need autonomously resolve unexpected clashes.

Modern AMR Platform



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

FOCUS ON FINITE TRACES FOR TASKS

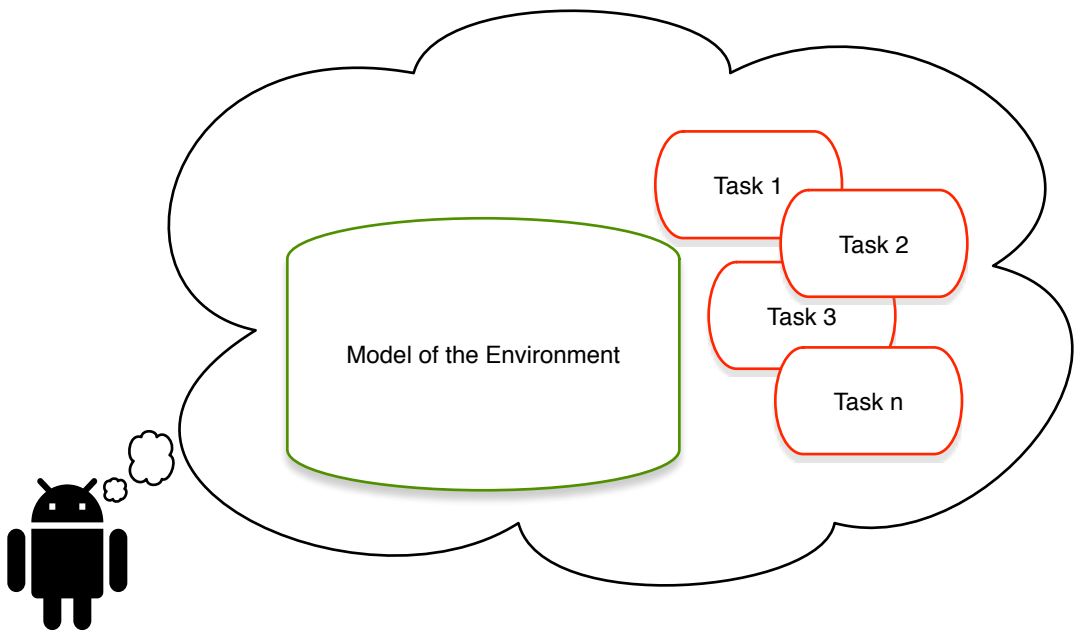
KR and Planning: Agent Tasks Must Terminate

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 of affair 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 models



Focus on Finite Traces for Tasks

In fact, the Reasoning about Actions and Planning community is adopting temporal logics since a long time often, interpret LTL on finite traces.

- Temporally extended goals [BacchusKabanza96] - infinite/finite
- Temporal constraints on trajectories [GereviniHaslumLongSaettiDimopoulos09 - PDDL3.0 2009] - finite
- Declarative control knowledge on trajectories [BaierMcIlraith06] - finite
- Procedural control knowledge on trajectories [BaierFrizMcIlraith07] - finite
- Temporal specification in planning domains [CalvaneseDeGiacomoVardi02] - infinite
- Planning via model checking - infinite
  - Branching time (CTL) [CimattiGiunchigliaGiunchigliaTraverso97]
  - Linear time (LTL) [DeGiacomoVardi99]

Finite traces also considered in Declarative Business Processes in Business Process Management [vanderAalstPesicSchonenberg2009]

Focus on Finite Traces for Tasks

$\mathrm{LTL}_f/\mathrm{LDL}_f$: linear temporal logics on finite traces [DeGiacomoVardi2013]

LTL_f: linear time temporal logic on finite traces

Same syntax as standard LTL but interpreted over finite traces

$$\varphi ::= A \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \mathit{next}\,\varphi \mid \mathit{eventually}\,\varphi \mid \mathit{always}\,\varphi \mid \varphi_1\ \mathit{until}\ \varphi_2$$

Examples:			
	$\mathit{eventually}\,A$	"eventually A"	*reachability*
	$\mathit{always}\,A$	"always A"	*safety*
	$\mathit{always}(A \rightarrow \mathit{eventually}\,B)$	"always if A then eventually B"	*reactiveness*
	$A\ \mathit{until}\ B$	"A until B"	*until*
	$\neg B\ \mathit{until}\ A \vee \mathit{always}\,\neg B$	"A before B"	*precedence*

LDL_f: linear dynamic logic on finite traces

Same syntax as PDL but interpreted over finite traces

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

Adds the possibility of expressing procedural constraints/goals [Reiter01], [BaierFritzMcIlraith07]:

$$\delta ::= A \mid \varphi? \mid \delta_1 + \delta_2 \mid \delta_1;\delta_2 \mid \delta^* \mid \textbf{if}\ \phi\ \textbf{then}\ \delta_1\ \textbf{else}\ \delta_2 \mid \textbf{while}\ \phi\ \textbf{do}\ \delta$$

where **if** and **while** are abbreviations: **if** ϕ **then** δ_1 **else** $\delta_2 \doteq (\phi?;\delta_1) + (\neg\phi?;\delta_2)$ and **while** ϕ **do** $\delta \doteq (\phi?;\delta)^*;\neg\phi?$

Focus on Finite Traces for Tasks

Example

- "*All coffee requests from person p will eventually be served*":

$$\textit{always}(request_p \rightarrow \textit{eventually}\, coffee_p) \qquad [\texttt{true}^*](request_p \rightarrow \langle \texttt{true}^* \rangle coffee_p)$$

- "*Every time the robot opens door d it closes it immediately after*":

$$\textit{always}(openDoor_d \rightarrow \textit{next}\, closeDoor_d) \qquad [\texttt{true}^*]([openDoor_d]closeDoor_d)$$

- "*Before entering restricted area a the robot must have permission for a*":

$$\neg inArea_a \;\textit{until}\; getPerm_a \vee \textit{always}\, \neg inArea_a \qquad \langle(\neg inArea_a)^*\rangle getPerm_a \vee [\texttt{true}^*]\neg inArea_a$$

- "*Each time the robot enters the restricted area a it must have a new permission for a*":

$$\langle(\neg inArea_a{}^*; getPerm_a; \neg inArea_a{}^*; inArea_a; inArea_a^*)^*; \neg inArea_a{}^*\rangle end$$

- "*At every point, if it is hot then, if the air-conditioning system is off, turn it on, else don't turn it off*":

$$[\texttt{true}^*]\langle \textbf{if}\ (hot)\ \textbf{then}\ \textbf{if}\ (\neg airOn)\ \textbf{then}\ turnOnAir\ \textbf{else}\ \neg turnOffAir\rangle true$$

Focus on Finite Traces for Tasks

Key point

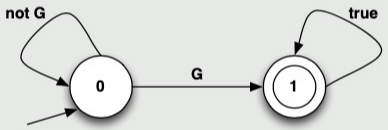
$\text{LTL}_f/\text{LDL}_f$ formulas can be translated into deterministic finite state automata (DFA).

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

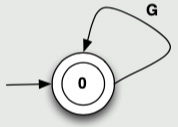
where A_φ is the DFA φ is translated into.

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

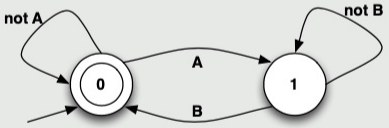
eventually G



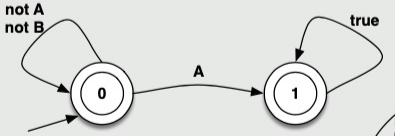
always G



always$(A \rightarrow$ *next eventually* $B)$

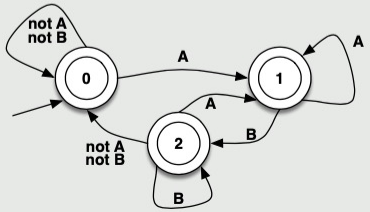


$\neg B$ *until* $A \vee$ *always* $\neg B$ *"A before B"*



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

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



Focus on Finite Traces for Tasks

FOND for LTL_f goals

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

1: Given a FOND domain $\mathcal{D}$ and an $\mathrm{LTL}_f/\mathrm{LDL}_f$ 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 $\mathrm{LTL}_f/\mathrm{LDL}_f$ goals is:

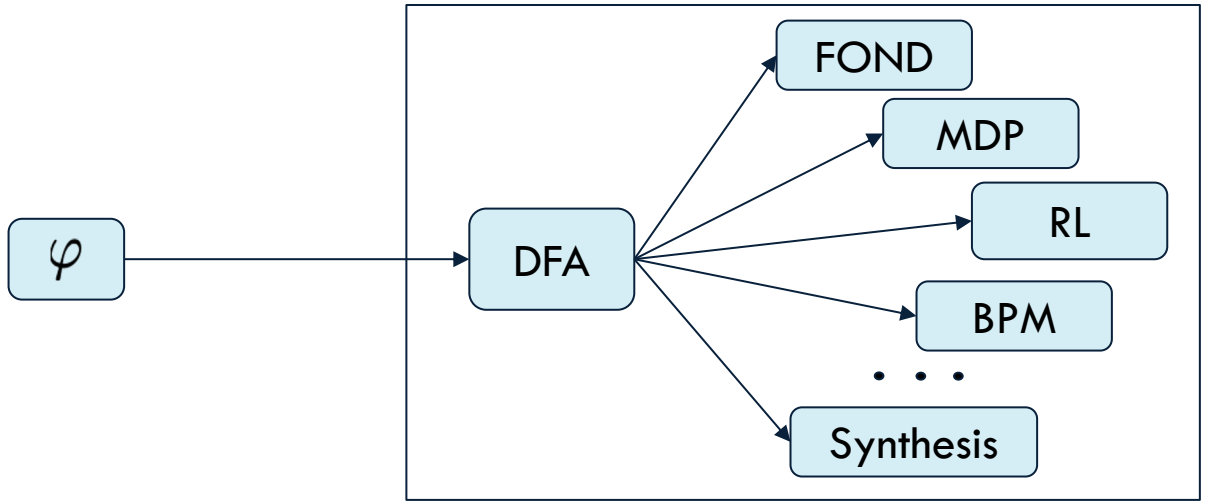
- *EXPTIME-complete in the domain (assuming a logarithmic representation as in PDDL);*
- *2EXPTIME-complete in the goal.*

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])

Focus on Finite Traces for Tasks

Many Applications:

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



SAPIENZA
UNIVERSITÀ DI ROMA

Reactive Synthesis, Planning and Reinforcement Learning in the Finite Trace Setting

MODELS OF THE ENVIRONMENT

Classic KR, Planning, MDPs Focuses on State-based Models

Classically in AI we focus on

state-based models of the environment:

- Environment is in a state
- Agent actions effects (and preconditions) depend only on the current state
- History of how we got in a certain state plays no role
- Action effects manifest at the very next state

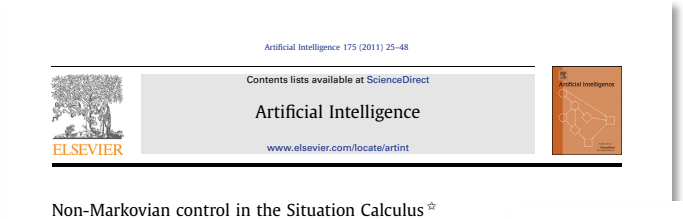
Beyond State-based Models

Though, this classic state-based view has been challenged many times!

For example:

- Non-Markovian action theories in Reasoning about Actions (e.g., in the Situation Calculus) [GabaldonAIJ2011]
  - Effects depend on the past history (safety properties)

- Trajectory constraints in Planning [CimattiPistoreRoveriTraversoAIJ2003]
  - Planning domain is a transition system/game arena
  - But in reacting to agent actions the environment has to fulfill certain temporal rules (originally forms of fairness)



Artificial Intelligence 175 (2011) 25–48

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

ELSEVIER

Non-Markovian control in the Situation Calculus ☆

Alfredo Gabaldon

Center for Artificial Intelligence, New University of Lisbon, Lisbon, Portugal

ARTICLE INFO

Article history:
Available online 3 April 2010

Keywords:
Reasoning about actions
Situation Calculus

ABSTRACT

In reasoning about actions, it is commonly assumed that
satisfies the *Markov Property*: the executability conditions and
are fully determined by the present state of the system. Th
Reiter's Basic Action Theories in the Situation Calculus. In
Basic Action Theories by removing the Markov property rest
to directly axiomatize actions whose effects and executability
past and even alternative, hypothetical situations. We then ge
operator, which is the main computational mechanism used for
Theories, so that it can be used with non-Markovian theories.
© 2010 Elsev

Since the 1960's when John McCarthy's papers (in particular the 1969 paper with Pat Hayes) a
Situation Calculus, researchers have been studying and working on this language for reasoning abou
Situation Calculus, one of John's many great inventions, is the topic of this paper and I am delighted
to make a contribution to a special issue in John's honor.

1. Introduction

An assumption commonly made in formalisms for reasoning about the effects of actions is the s
the executability of an action and its effects are entirely determined by the current state or situati
Basic Action Theories [2], a Situation Calculus [3,4] based axiomatization, define the value of a fl
of an action in terms of a formula that can only talk about the situation in which the action
preconditions of an action are specified by formulas with the same restriction. In this paper we
Theories by removing this restriction. The generalized theories will allow the executability conditio
action to depend not only on what holds when the action is to occur, but also on whether certain
at different points in the past and even alternative hypothetical evolutions of the system.
As an example, imagine a robot that works in a biological research facility with different safety-l
is such that a material will be considered contaminated after the robot touches it if the robot has b
or has directly been in contact with a hazardous material, and has not been to the disinfection st
effect of touching the material depends on the history of robot activities. We could also imagin
execute the action *open*(*Entrance*, *Lab*1) if *temp*(*Lab*1) > 30 was ever true since the last time *closed*(
The latter is an example of an action with non-Markovian preconditions.
In simple scenarios, it is not difficult to extend a theory to preserve the necessary history
variables, especially when the domain is finite. But in complex domains it may not be obvious

☆ A preliminary abstract of this paper appeared in Proc. of AAAI'02 (A. Gabaldon (2002) [1]).
E-mail address: ag@di.fct.unl.pt.

0004-3702/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.artint.2010.04.012

ELSEVIER

Available at www.ComputerScienceWeb.com
POWERED BY BOISHOR@DIRBOT

Artificial Intelligence 147 (2003) 35–84

Artificial Intelligence

www.elsevier.com/locate/artint

Weak, strong, and strong cyclic planning via symbolic model checking

A. Cimatti *, M. Pistore, M. Roveri, P. Traverso

ITC-IRST, Via Sommarive 18, 38055 Povo, Trento, Italy

Received 22 June 2001; received in revised form 3 May 2002

Abstract

Planning in nondeterministic domains yields both conceptual and practical difficulties. From the conceptual point of view, different notions of planning problems can be devised: for instance, a plan might either guarantee goal achievement, or just have some chances of success. From the practical point of view, the problem is to devise algorithms that can effectively deal with large state spaces. In this paper, we tackle planning in nondeterministic domains by addressing conceptual and practical problems. We formally characterize different planning problems, where solutions have a chance of success ("weak planning"), are guaranteed to achieve the goal ("strong planning"), or achieve the goal with iterative trial-and-error strategies ("strong cyclic planning"). In strong cyclic planning, all the executions associated with the solution plan always have a possibility of terminating and, when they do, they are guaranteed to achieve the goal. We present planning algorithms for these problem classes, and prove that they are correct and complete. We implement the algorithms in the MBP planner by using symbolic model checking techniques. We show that our approach is practical with an extensive experimental evaluation: MBP compares positively with state-of-the-art planners, both in terms of expressiveness and in terms of performance.
© 2003 Elsevier Science B.V. All rights reserved.

Keywords: Planning in nondeterministic domains; Conditional planning; Symbolic model-checking; Binary decision diagrams

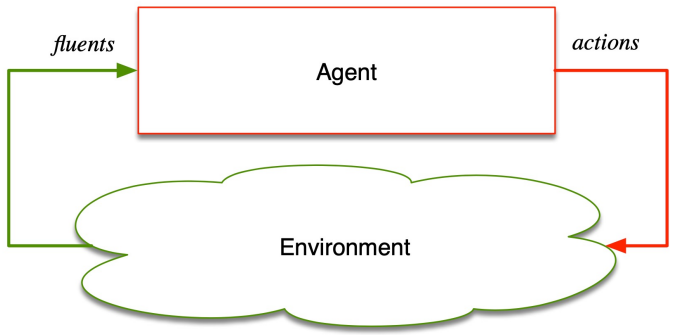
* Corresponding author.
E-mail addresses: cimatti@irst.itc.it (A. Cimatti), pistore@irst.itc.it (M. Pistore), roveri@irst.itc.it (M. Roveri), traverso@irst.itc.it (P. Traverso).

0004-3702/03/$ – see front matter © 2003 Elsevier Science B.V. All rights reserved.
doi:10.1016/S0004-3702(02)00374-0

Models of Environment Must Capture its Possible Behaviors!

Agent and environment behaviors must be processes!

- Define **alphabet**
  - of actions for the agent
  - of fluents for the environment
- Behaviors (aka strategies/policies/protocols/plans) must be **processes** [AbadiLamportWolper89]
  - Functions that chooses the next move on the base of the history so far.



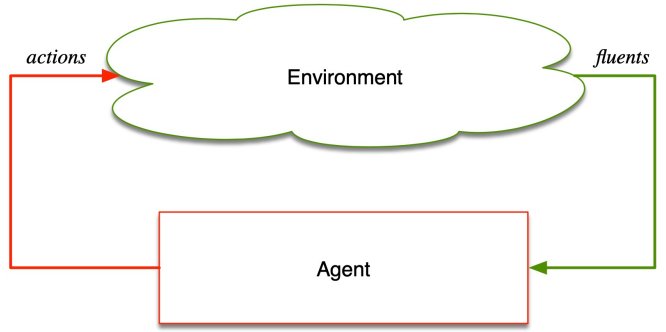
Agent Behavior

$$\sigma_a : (fluents)^* \rightarrow actions$$

where

- $(fluents)^*$ denotes the **history** of what observed so far by the agent
  (a finite sequence of fluents configurations)
- $actions$ denotes the **next action** that the agent does

Every program/process has this form! [AbadiLamportWolper89].



Environment Behavior

$$\sigma_e : (actions)^* \rightarrow fluents$$

where

- $(actions)^*$ denotes the **history** of what observed so far by the environment
  (a finite sequence of agent actions)
- $fluents$ denotes the **next effects** that the environment brings about.

Every program/process has this form! [AbadiLamportWolper89].

Reasoning about Actions and Planning: Domains as Environment Specifications

Domain

- Planning considers the agent acting in a **(nondeterministic) domain**
- The domain is a **model of how the environment** works
- That is, it is a **specification of the possible environment behaviors**

$$[\![Dom]\!] = \{\sigma_e | \sigma_e \text{ compliant with } Dom\}$$

The presence of domain is a crucial point of planning since the beginning!

Planning in nondeterministic domains

Given an LTL_f task $Goal$ for the agent, and a domain Dom modeling the environment

Find agent behavior σ_a such that $\forall \sigma_e \in [\![Dom]\!].trace(\sigma_a, \sigma_e) \models Goal$

Reasoning about Actions and Planning: General LTL Properties as Environment Specifications

We can we use **LTL/LTLf specify the environment**, through the notion of **realizability**

Environment specifications in LTL

Let Env be an LTL/LTL_f formula over action and fluents.

$$[\![Env]\!] = \{\sigma_e | \forall \sigma_a . trace(\sigma_a, \sigma_e) \models Env\}$$

i.e Env denotes all environment behaviors that play according to the specification whatever is the agent behavior.

Synthesis with environment model in LTL/LTL_f

Given an LTL/LTL_f task $Task$ for the agent, and an LTL/LTL_f environment specification Env:

Find agent behavior σ_a such that $\forall \sigma_e \in [\![Env]\!] . trace(\sigma_a, \sigma_e) \models Task$

Reasoning about Actions and Planning: General LTL Properties as Environment Specifications

$\mathrm{LTL}/\mathrm{LTL}_f$ for modeling the environment

Many extensions!

1. Nondeterministic planning domains
2. Nondeterministic planning domains with non-Markovian effects [BrafmanDeGiacomoIJCAI2019], possibly referring to the past [DeGiacomoDiStasioFuggittiRubinIJCAI2020]
3. Forms fairness (*always eventually* ϕ), stability (*eventually always* ϕ), GR(1) (*always eventually* $\phi_1 \implies$ *always eventually* ϕ_2), and more general assumptions [ZhuDeGiacomoPuVardiAAAI2020], [AminofDeGiacomoRubinICAPS2020], [DeGiacomoDiStasioVardiZhuKR2020],[DeGiacomoDiStasioTabajaraVardiZhuKR2021sub]
4. Mandatory $stop$ and safety goals [DeGiacomoDiStasioPerelliZhu2021sub]
5. Trajectory constraints on domain used in generalized planning (e.g., *always eventually* $dec \rightarrow$ *eventually* $x = 0$) [BonetDeGiacomoGeffnerRubinIJCAI2017], [BonetGeffnerDeGiacomoPatriziRubinKR2020]

SAPIENZA UNIVERSITÀ DI ROMA

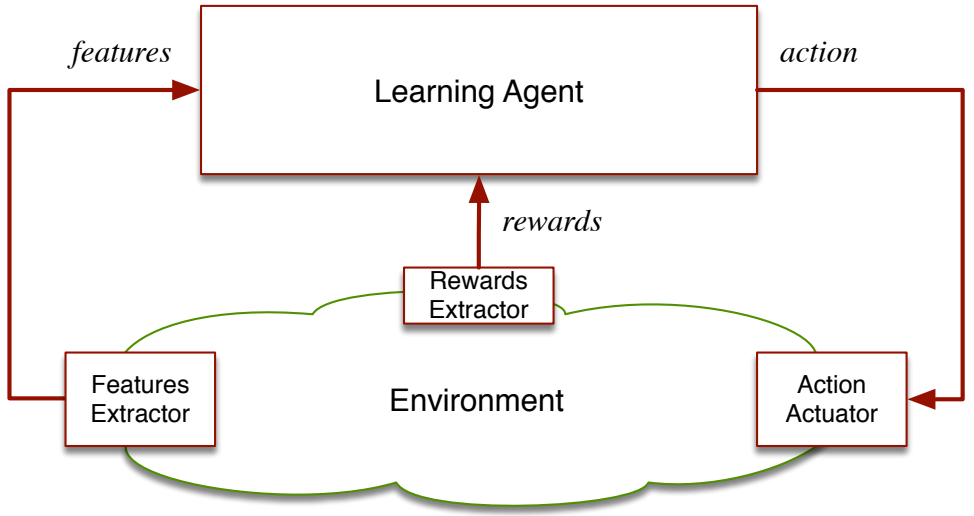
Restraining Bolts for Reinforcement Learning Agents

MERGING REASONING AND LEARNING

Learning Agents and Reasoning Agents

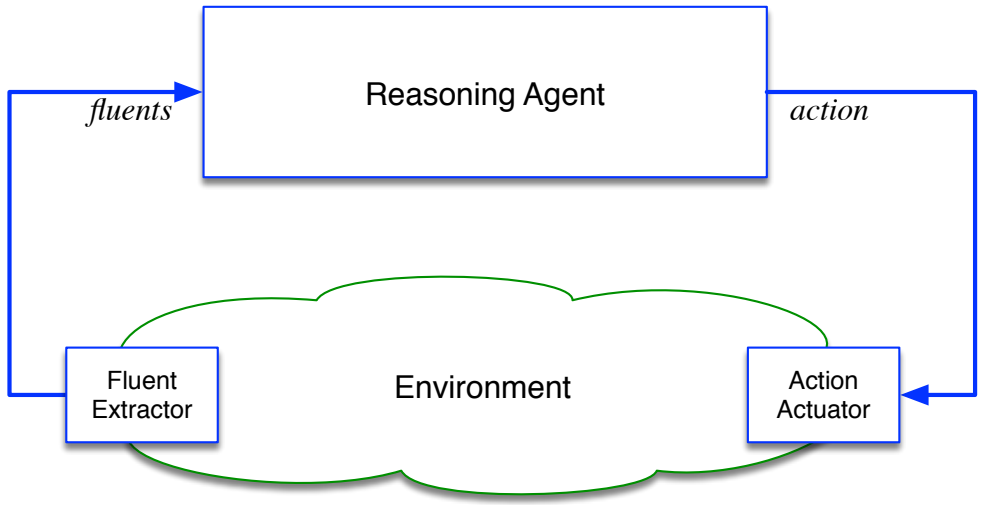
Learning agent:

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



Reasoning agent:

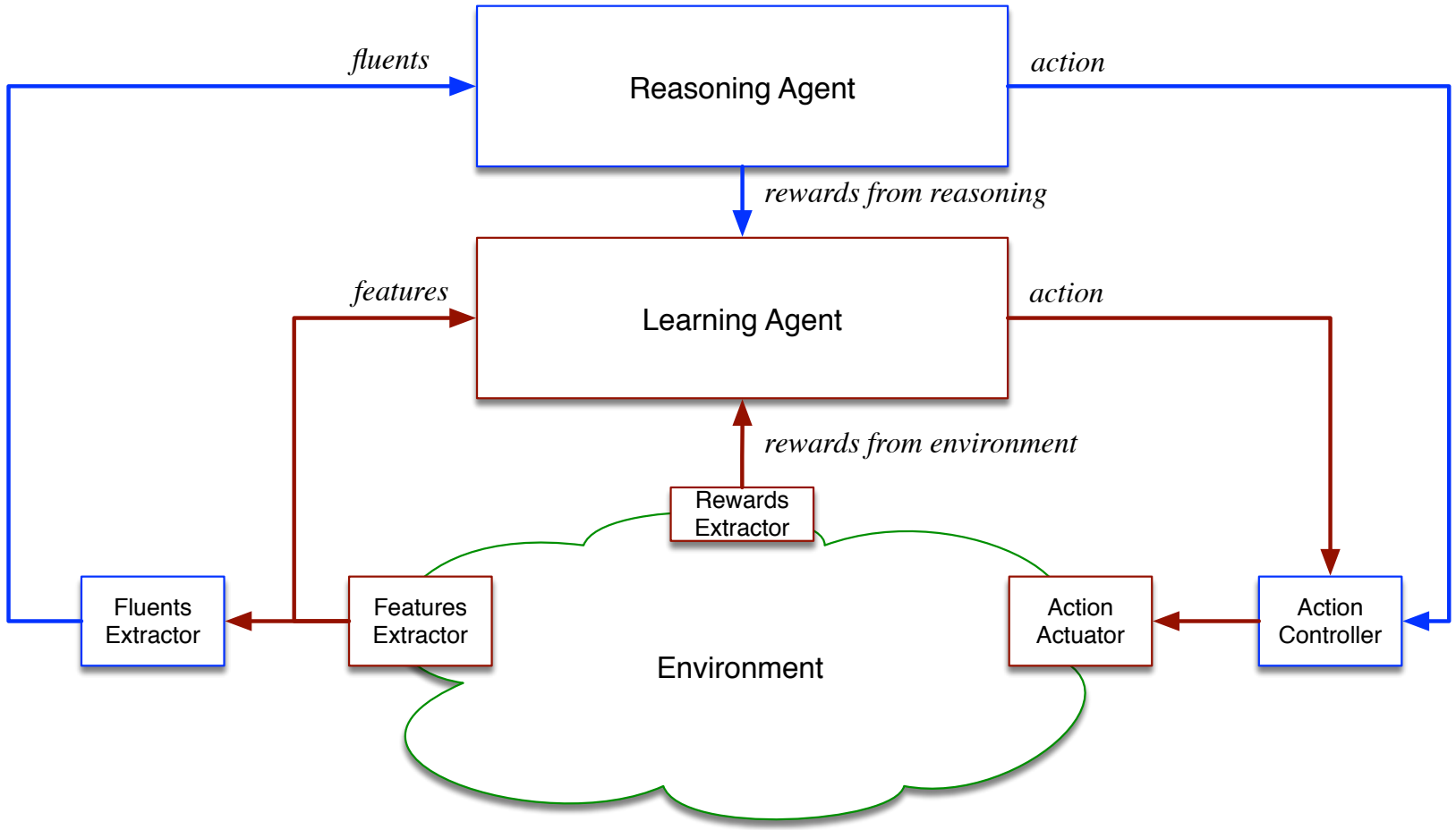
- Senses and acts on the environment
- Has models of its environment and tasks
- Does reasoning and planning



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



MDPs with Logic-based non-Markovian Rewards

MDPs with non-Markovian rewards

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

  MDP without rewards

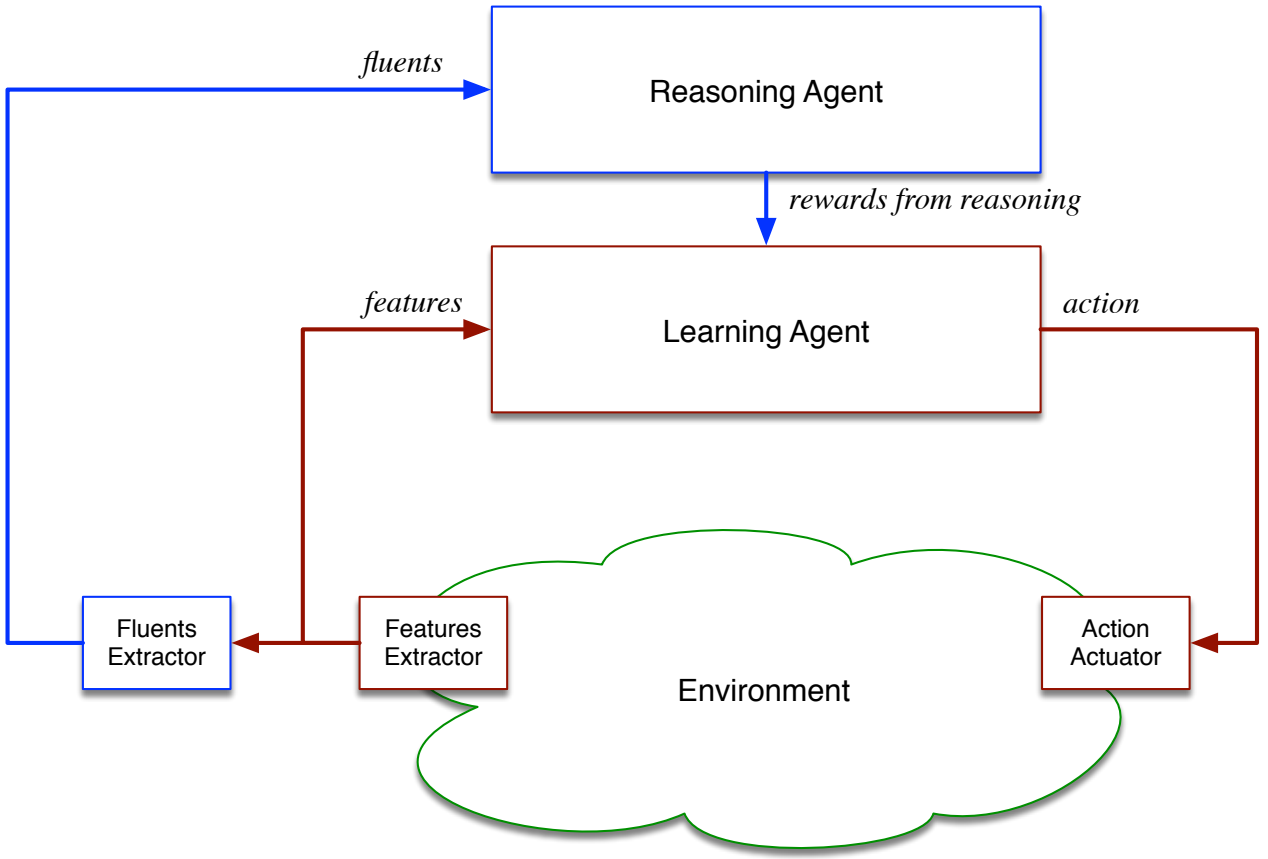
- Reasoning agent: $\mathcal{R} = (\mathcal{L}, \{(\varphi_i, r_i)\}_{i=1}^{m})$

  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



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 as Reasoning Agents

- Given an NMRDP $\mathcal{N} = \langle S, A, Tr, \{(\varphi_i, r_i)\}_{i=1}^{m} \rangle$ with:
  - rewards r_i specified by $\mathrm{LTL}_f/\mathrm{LDL}_f$ formulas φ_i
  - over a set of propositional symbols $\mathcal{P}$
  - agent's state space: $S \subseteq 2^{\mathcal{P}}$

- We can transform it into an equivalent MDP $\mathcal{M} = \langle S', A, Tr', R' \rangle$ with:
  - $S' = Q_1 \times \cdots \times Q_m \times S$ where Q_i is the set of states of DFAs $\mathcal{A}_{\varphi_i}$
    i.e., the states of $\mathcal{M}$ are extended to include the states of (the DFAs for) the formulas
  - $Tr' : S' \times A' \times S' \to [0, 1]$ is defined as:

$$Tr'(q_1, \ldots, q_m, s, a, q'_1, \ldots, q'_m, s') = \begin{cases} Tr(s, a, s') & \text{if } \forall i : \delta_i(q_i, s') = q'_i \\ 0 & \text{otherwise;} \end{cases}$$

  - $R' : S' \times A \times S' \to \mathbb{R}$ is defined as:

$$R'(q_1, \ldots, q_m, s, a, q'_1, \ldots, q'_m, s') = \sum_{i : q'_i \in F_i} r_i$$

    i.e., reward is given iff q_i is an accepting state of $\mathcal{A}_{\varphi_i}$

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

G. De Giacomo, M. Favorito, L. Iocchi, F. Patrizi, A. Ronca. Temporal Logic Monitoring Rewards via Transducers. KR 2020.

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

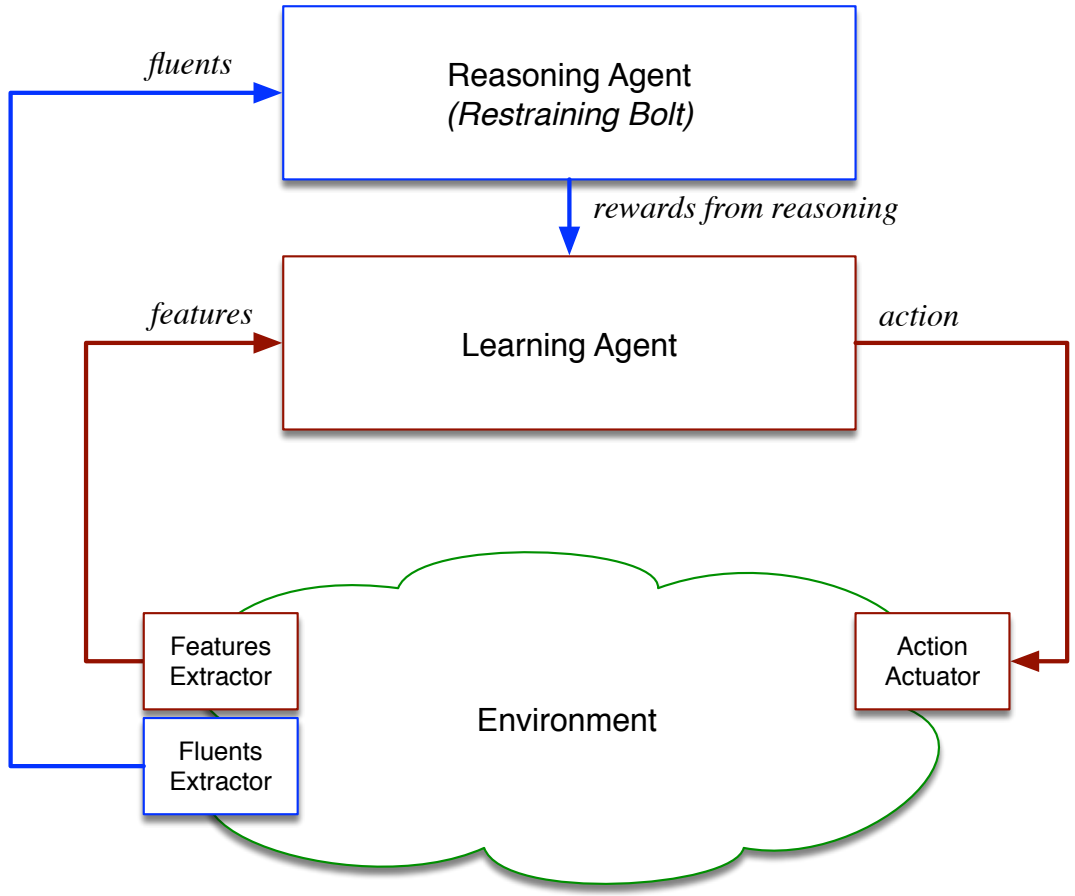
- Reasoning agent: $\mathcal{R} = (\mathcal{L}, \{(\varphi_i, r_i)\}_{i=1}^{m})$

  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.

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

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

RL with $\text{LTL}_f/\text{LDL}_f$ restraining specifications for learning agent $M = \langle S, A, Tr_{ag}, R_{ag} \rangle$ and restraining bolt $RB = \langle \mathcal{L}, \{(\varphi_i, r_i)\}_{i=1}^{m} \rangle$

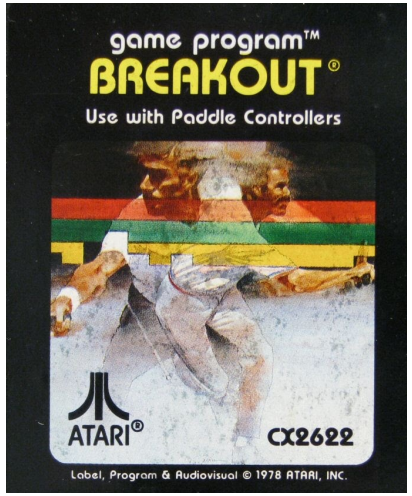
- **Transform each φ_i into** DFA $\mathcal{A}_{\varphi_i} = \langle 2^{\mathcal{L}}, Q_i, q_{io}, \delta_i, F_i \rangle$ over fluents evaluations $\mathcal{L}$ with states Q_i and final states $F_i \subseteq Q_i$.
- **Do classical RL over a new MDP $M' = \langle Q_1 \times \cdots \times Q_m \times S, A, Tr'_{ag}, R'_{ag} \rangle$**
- **Thm: the optimal policy ρ'_{ag} learned for M' is an optimal policy of the original problem.**[a]

[a] Crux of the result: the reward function depends on features and automata states, not on fluents

$$R'_{ag}(q_1, \ldots, q_m, s, a, q'_1, \ldots, q'_m, s') = \sum_{i : q'_i \in F_i} r_i + R_{ag}(s, a, s')$$

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

Example: Breakout

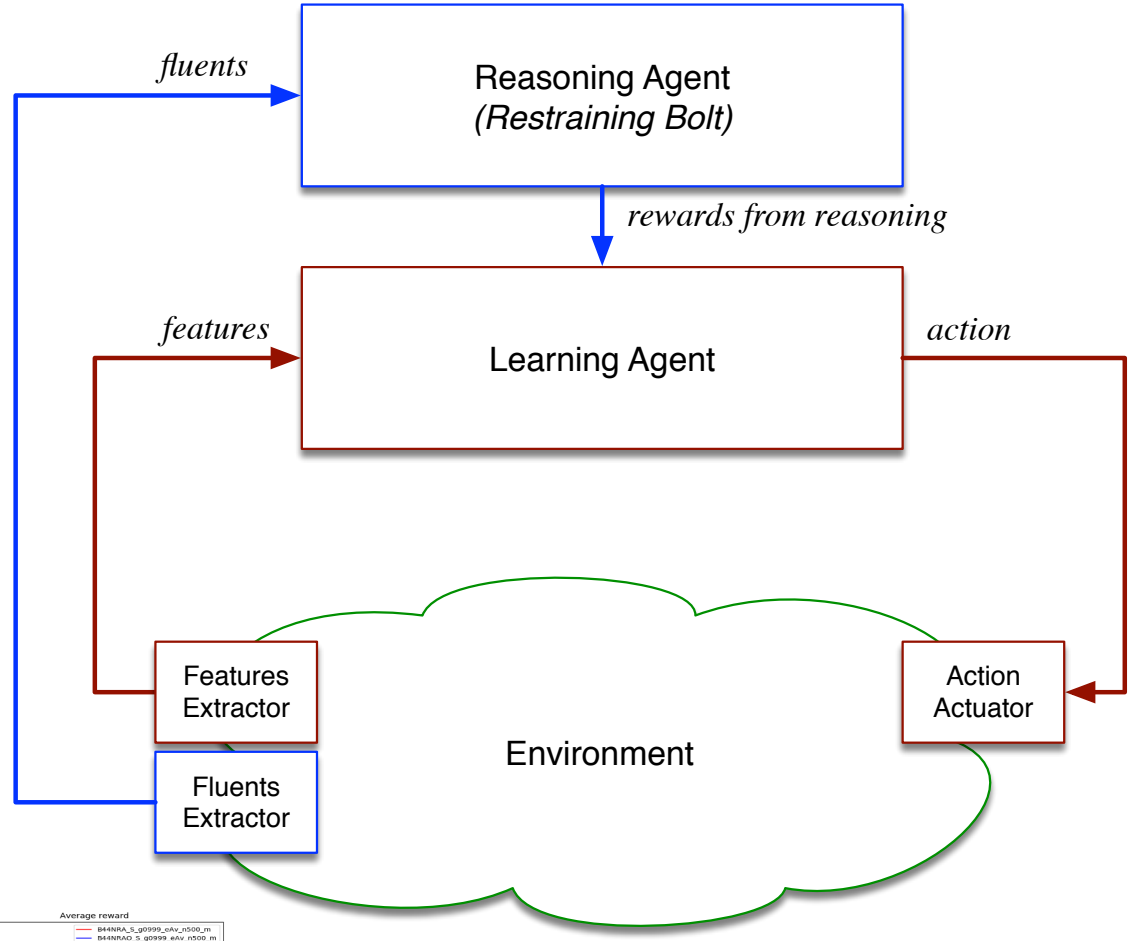


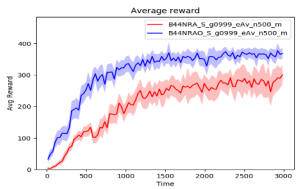
Learning Agent

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

Restraining Bolt (Reasoning Agent)

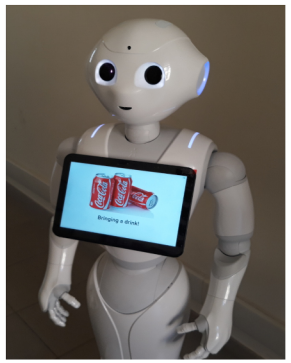
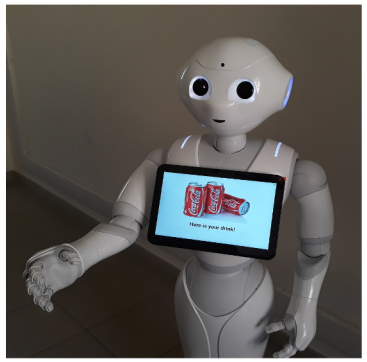
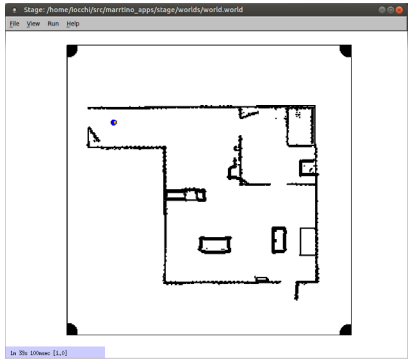
- Rewards: break one column at the time left to right (all bricks in column i must be removed before completing any other column j > i)
- Fluents: bricks/columns status (broken/not broken)





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

Example: Cocktail Party

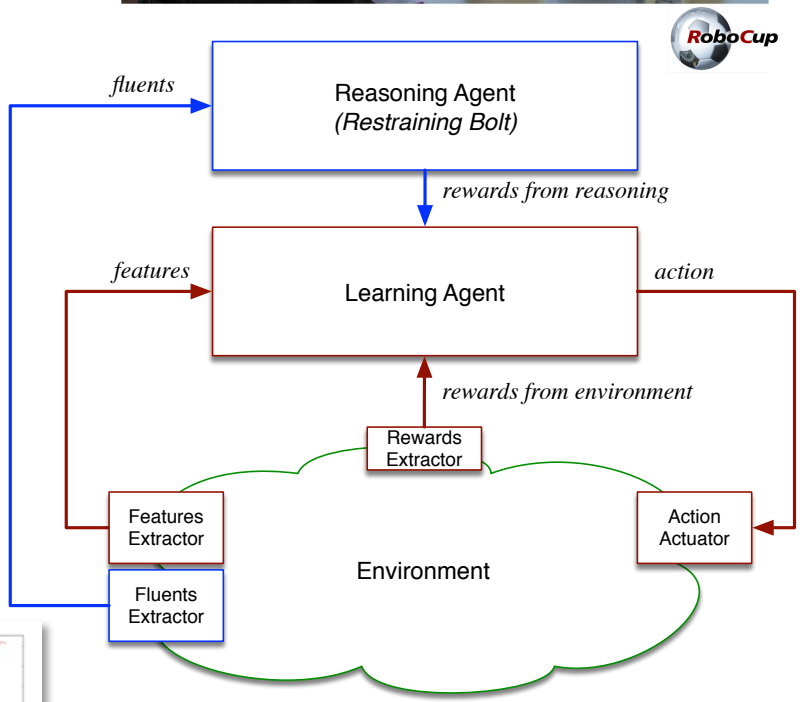





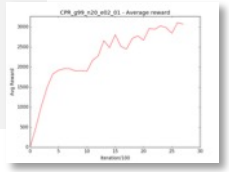
Learning Agent

- 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)

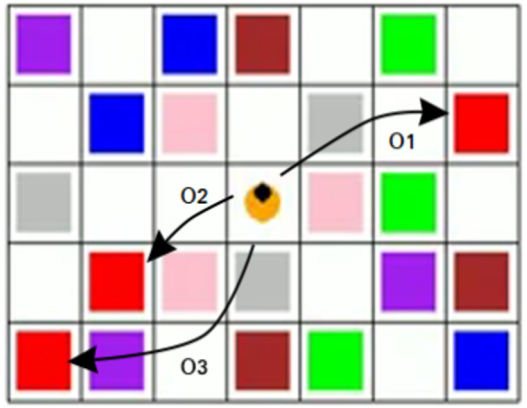
- Rewards: serve exactly one drink and one snack to every person, and do not serve alcoholic drinks to minors
- Fluents: identity and age of people, and received items *(uses Microsoft Cognitive Services Face API to provide information)*





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

Example: Sapientino

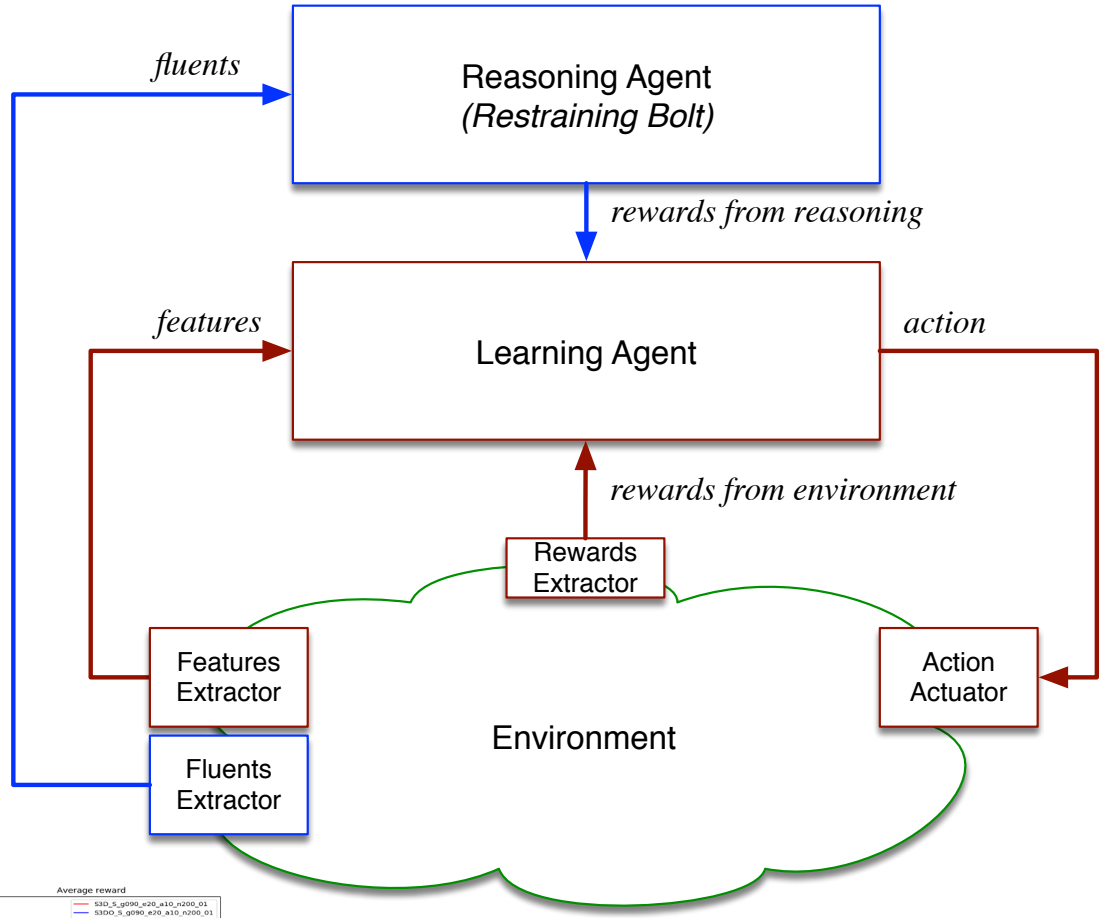


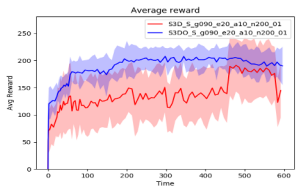
Learning Agent

- Features: robot position (x, y) and facing θ
- Actions: forward, backward, turn left, turn right, beep
- Rewards: negative rewards when agent exits board

Restraining Bolt (Reasoning Agent)

- Rewards: visit (beep) at a sequence of three positions of the same colour for each colour
- Fluents: colour of current cell, just beeped

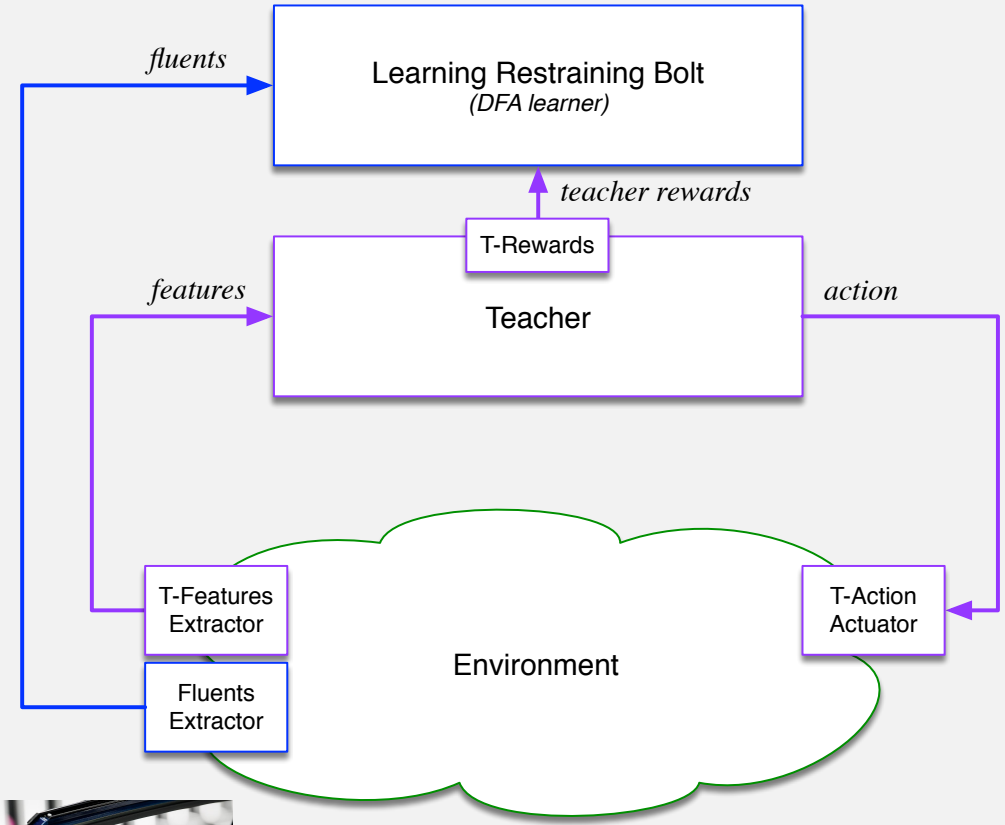




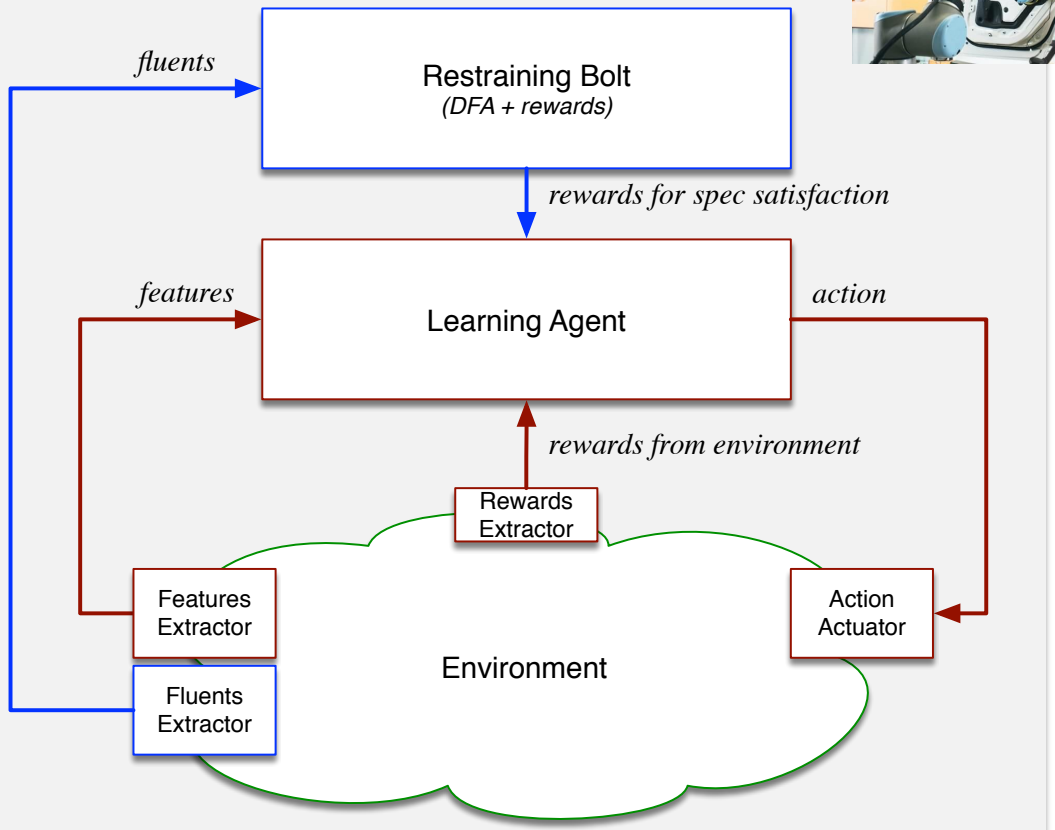
https://sites.google.com/diag.uniroma1.it/restraining-bolt

Extensions: Imitation Learning

Learn a Restraining Bolt



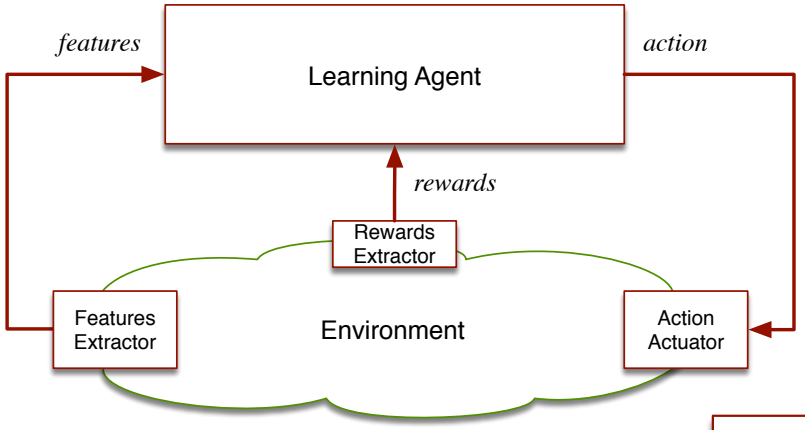
Use the Restraining Bolt

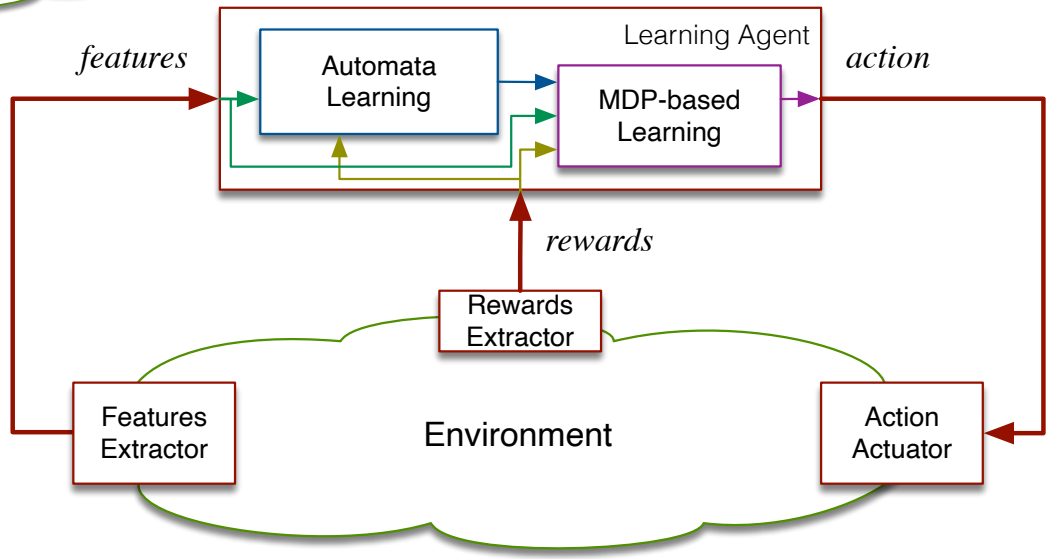


G. De Giacomo, M. Favorito, L. Iocchi, and F. Patrizi.
Imitation Learning over Heterogeneous Agents with Restraining Bolts. ICAPS 2020.
https://whitemech.github.io/Imitation-Learning-over-Heterogeneous-Agents-with-Restraining-Bolts

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.

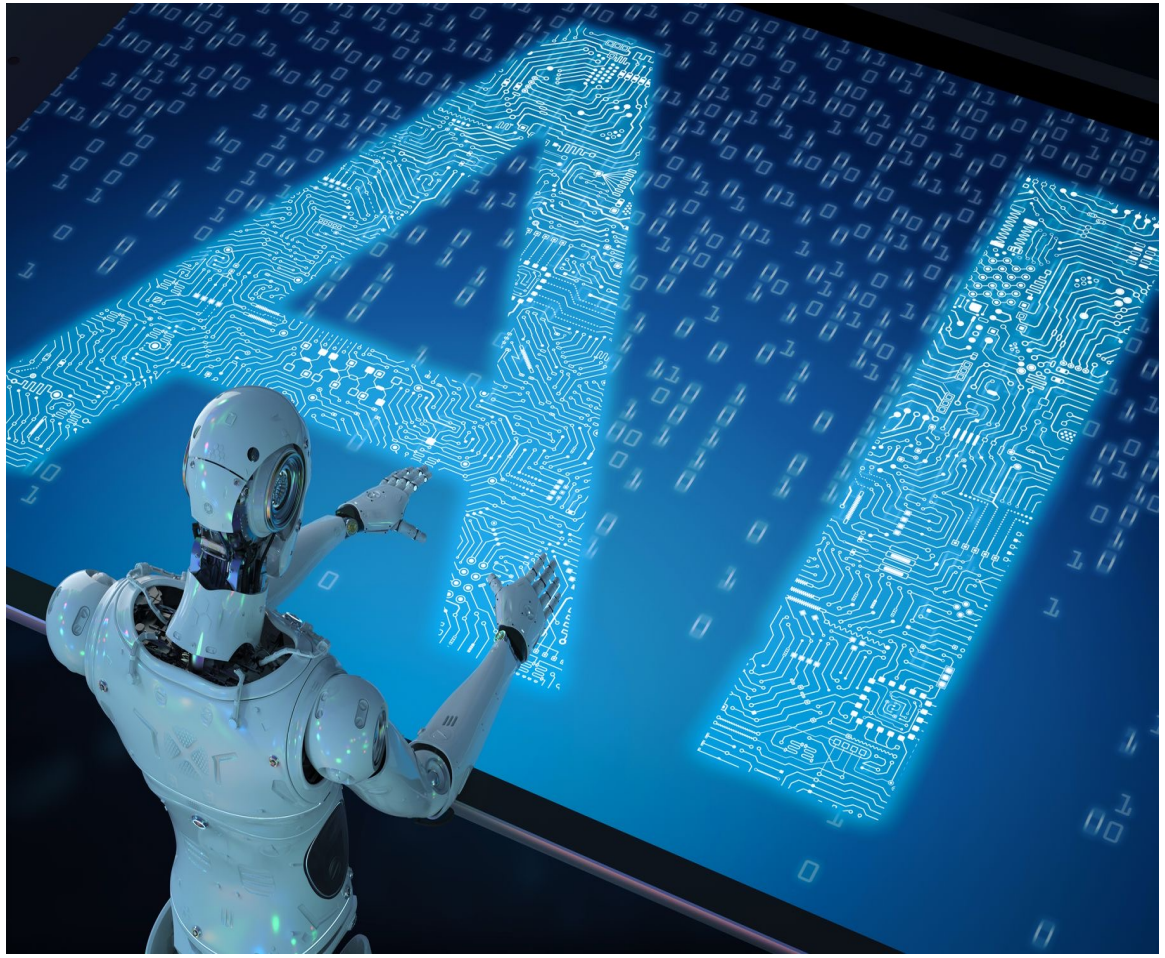
A. Ronca, G. De Giacomo. Efficient PAC Reinforcement Learning in Regular Decision Processes. IJCAI 2021.

Restraining Bolts for Reinforcement Learning Agents

CONCLUSIONS

Conclusions

- Autonomy 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
- Merging reasoning and learning is one of the most important challenges for autonomy in AI. Encouraging results are available
- One more thing. Goal Formation (where do the goal come from?) related to Goal Reasoning: obedient agents, rebellious agents, and agents that change mind through interaction.



G. De Giacomo, Y. Lesperance. Goal Formation through Interaction in the Situation Calculus: A Formal Account Grounded in Behavioural Science. AAMAS 2020. *(talk available on underline.io)*