Logic, Automata, and Games in Linear Temporal Logics on Finite Traces

Giuseppe De Giacomo

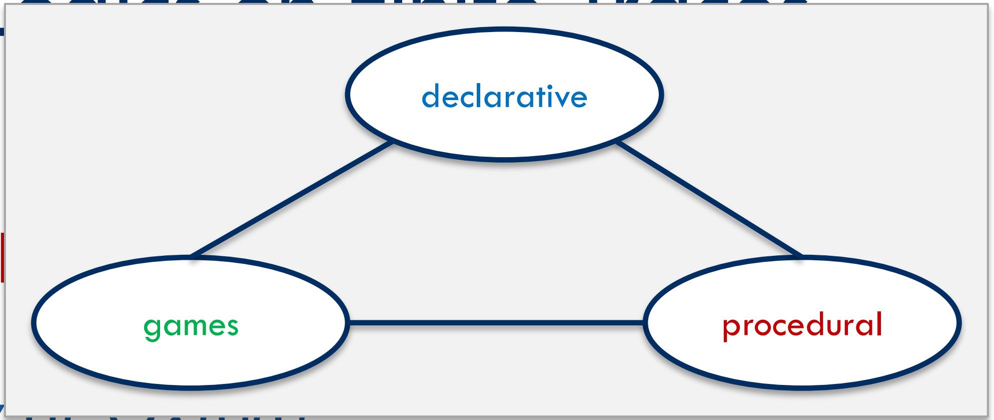
University of Oxford

GenPlan@NeurIPS2023 – New Orleans, USA
December 16, 2023

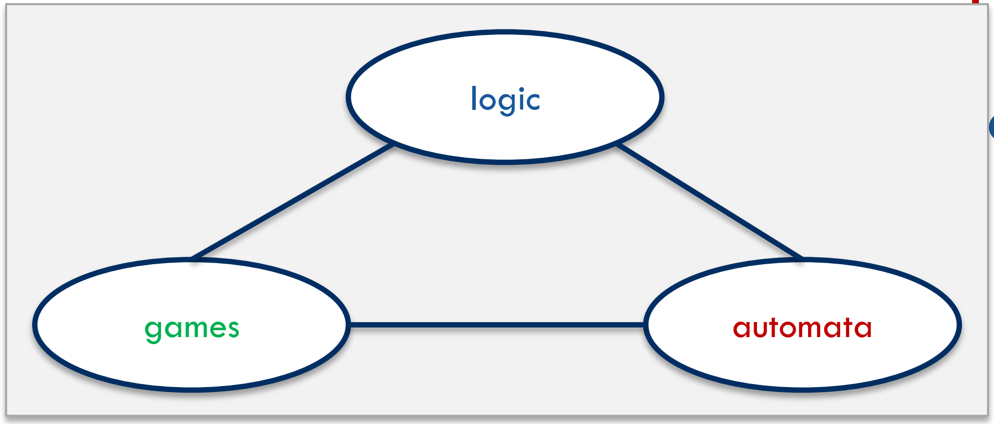
TAILOR https://tailor-network.eu/

UNIVERSITY OF OXFORD

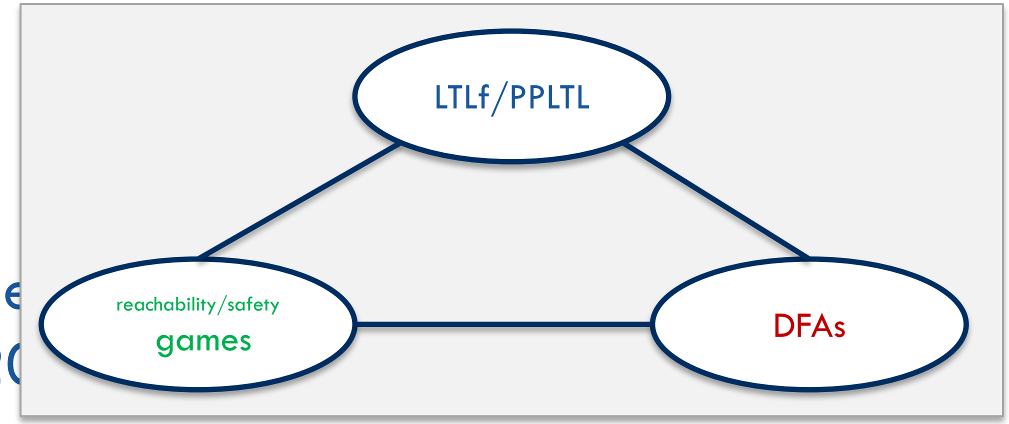
Logic, Automata, and Games
in Linear Temporal L



Giuseppe

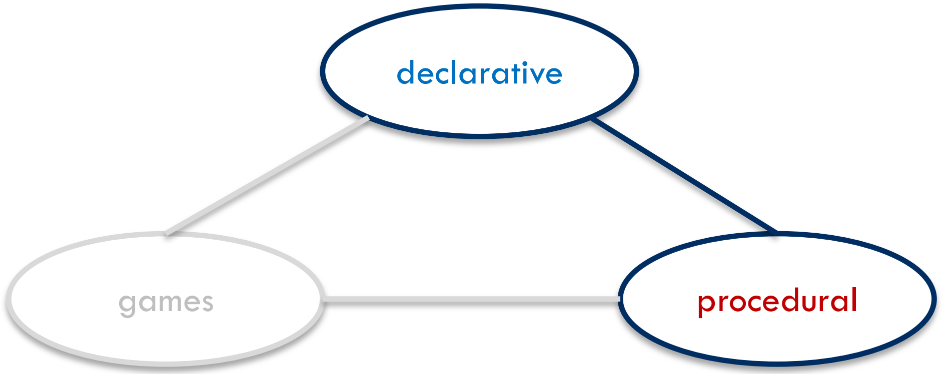


ersity

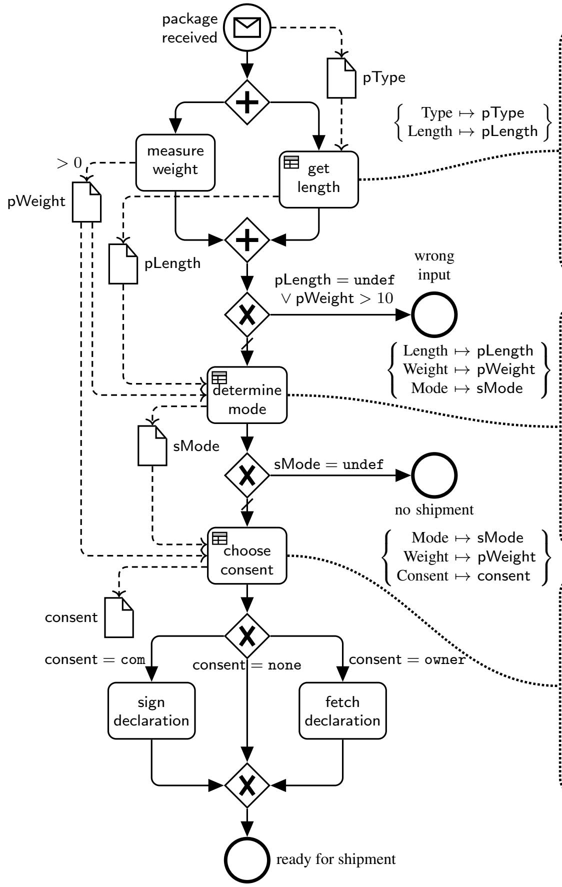


GenPlan@NeurIPS2023 – Ne
December 16, 20

DECLARATIVE …
… TO PROCEDURAL



Procedural Specs



Get Length

U	Type	Length (m)
	`std,large,xl`	> 0
1	`std`	0.5
2	`large`	1
3	`xl`	2

Determine Mode

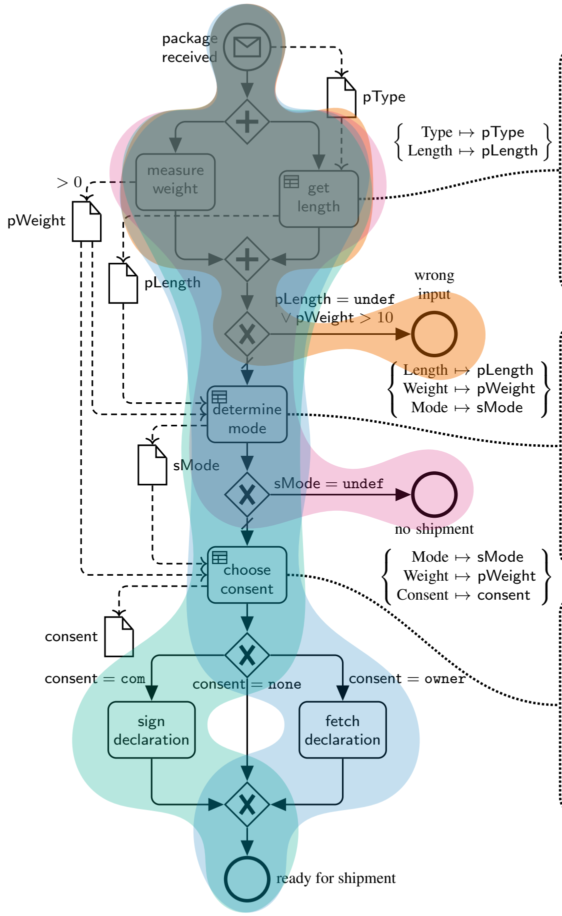
U	Length (m)	Weight (kg)	Mode
	> 0	> 0	`car, truck`
1	$(0, 1]$	$(0, 5]$	`car`
2	$(1, 2]$	$(0, 5]$	`truck`
3	–	$(5, 10]$	`truck`

Choose Consent

U	Mode	Weight (kg)	Consent
	`car,truck`	> 0	`none`, `owner`, `com`
1	`car`	> 6	`owner`
2	`truck`	> 8	`com`

Courtesy of Marco Montali

Procedural Specs Induce Traces



Get Length

U	Type	Length (m)
	std,large,xl	> 0
1	std	0.5
2	large	1
3	xl	2

Determine Mode

U	Length (m)	Weight (kg)	Mode
	> 0	> 0	car, truck
1	(0, 1]	(0, 5]	car
2	(1, 2]	(0, 5]	truck
3	–	(5, 10]	truck

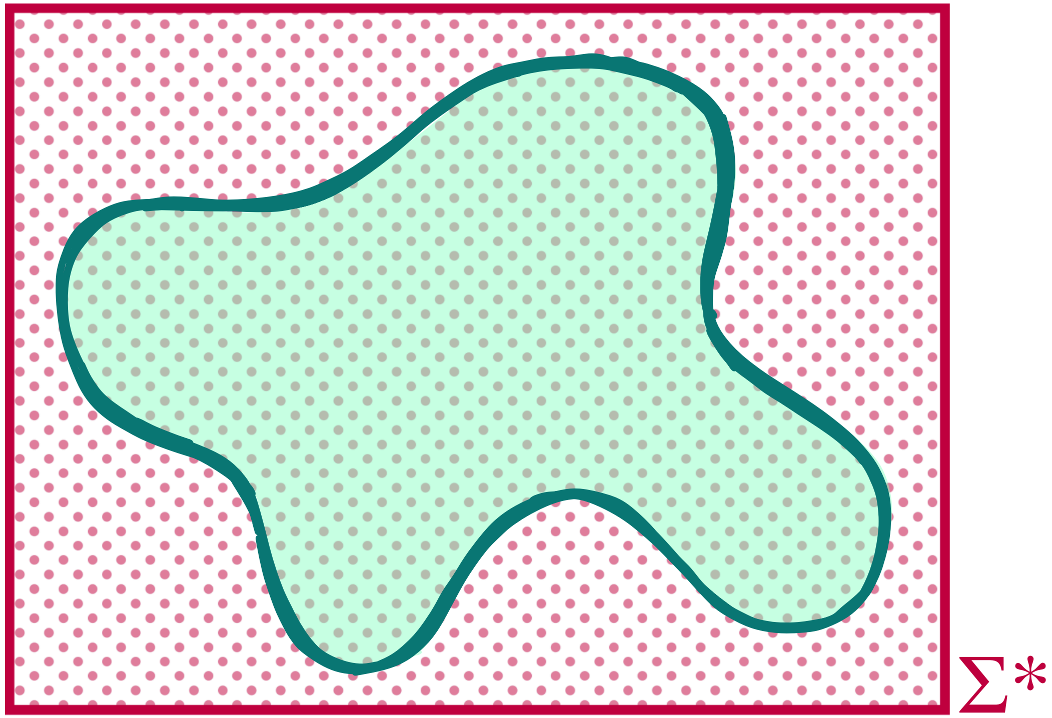
Choose Consent

U	Mode	Weight (kg)	Consent
	car,truck	> 0	none, owner, com
1	car	> 6	owner
2	truck	> 8	com

Courtesy of Marco Montali

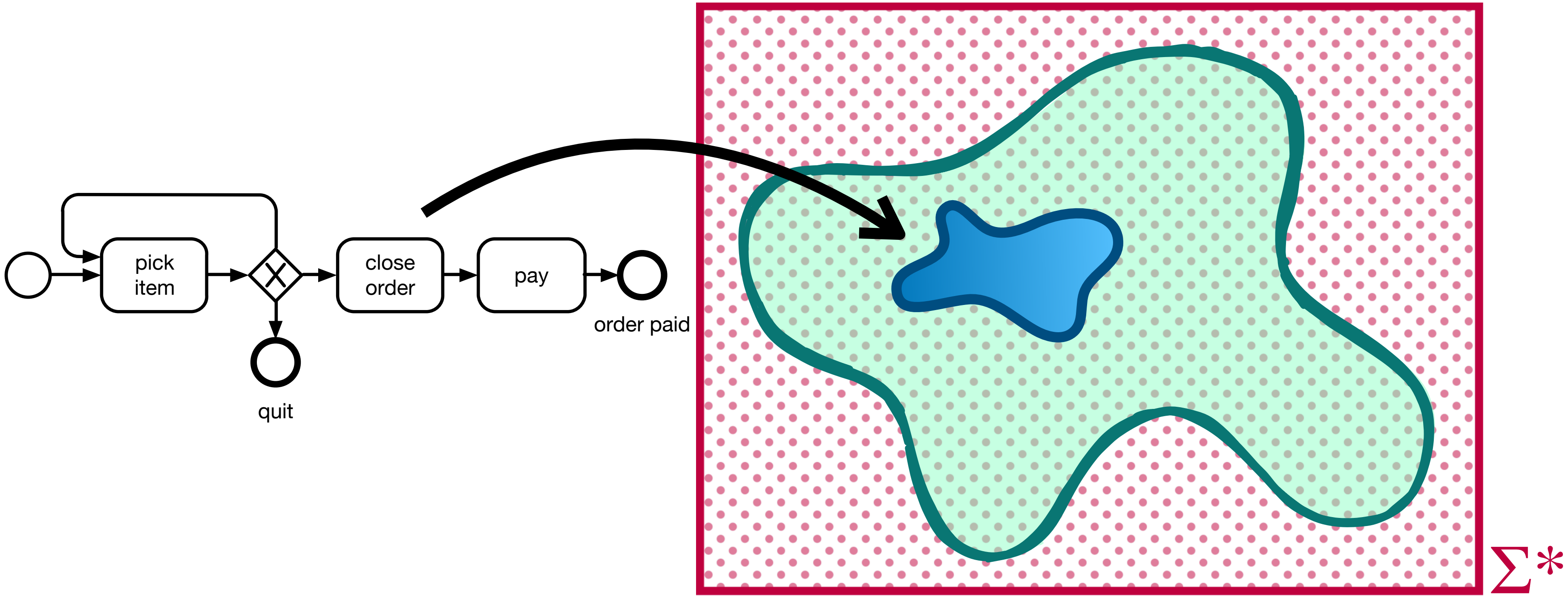
Traces Allowed by the Specs

A possibly <u>infinite set</u> of <u>finite traces</u>



Courtesy of Marco Montali

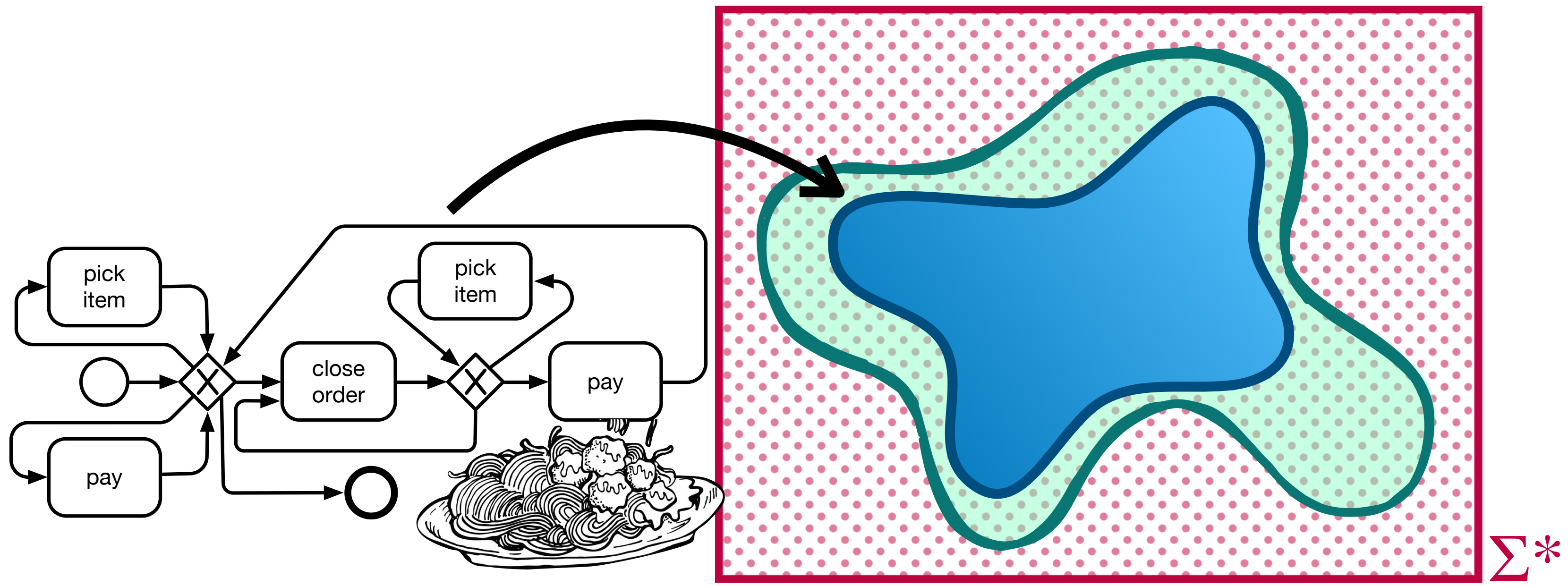
Traces Allowed by the Specs



Courtesy of Marco Montali

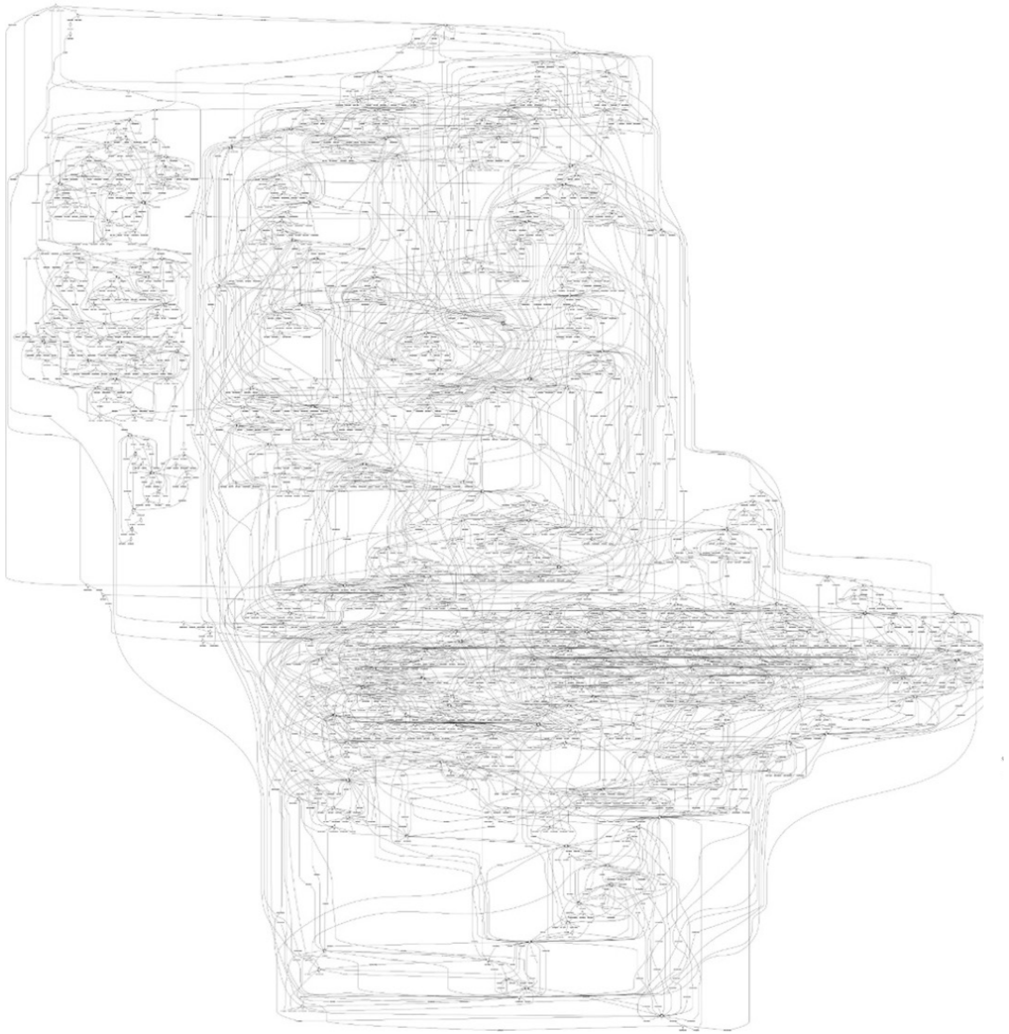
Traces Allowed by the Specs

Generalisation



Courtesy of Marco Montali

Reality Is Often More Flexible Than It Seems



Courtesy of Marco Montali

Declarative (Specifications of) Processes

Our goal

represents

Compact specification

Reality

Courtesy of Marco Montali

Declarative Processes

In late 2000's the Business Process Management (BPM) community produced a brilliant idea:
[PesicVanDerAalst06] [AlbertiEtAlt06] [Montali2010] [PesicBovsnavkiVanDerAalst10]

The idea:

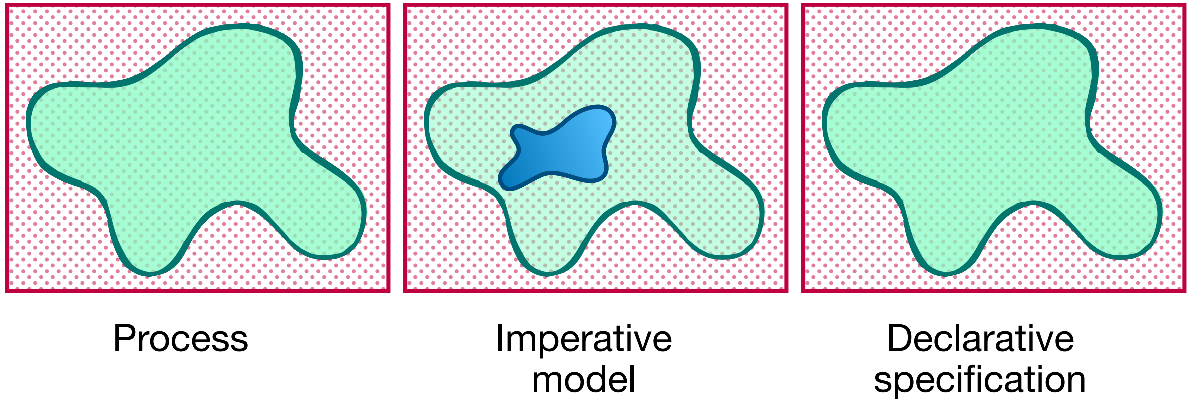
- Give the rules that a process should satisfy
- And nothing else!
- Extract the process from the rules only

Which specs?

- The one most used in formal methods for specifying process properties
- Linear Time Logic on Finite Traces (LTLf)

In other words:

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



Declarative Processes

In late 2000's the Business Process Management (BPM) community produced a brilliant idea: [PesicVanDerAalst06] [AlbertiEtAlt06] [Montali2010] [PesicBovsnavkiVanDerAalst10]

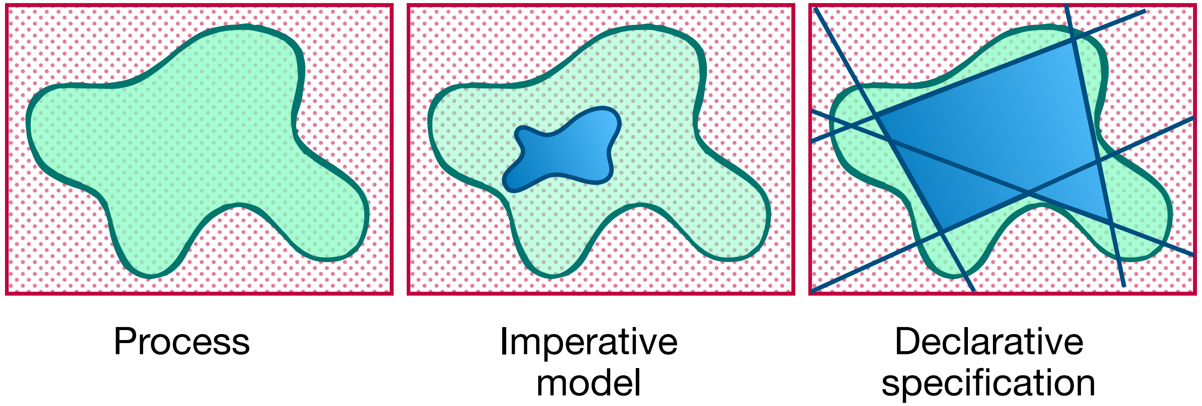
The idea:

- 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)



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	$\boxed{a}^{1..*}$	$\Diamond a$	a must be executed at least once
Resp. existence	$\boxed{a}$•—$\boxed{b}$	$\Diamond a \supset \Diamond b$	If a is executed, then b must be executed as well
Response	$\boxed{a}$•—▶$\boxed{b}$	$\Box(a \supset \Diamond b)$	Every time a is executed, b must be executed afterwards
Precedence	$\boxed{a}$—▶•$\boxed{b}$	$\neg b \mathcal{W} a$	b can be executed only if a has been executed before
Alt. Response	$\boxed{a}$•═▶$\boxed{b}$	$\Box(a \supset \bigcirc(\neg a \mathcal{U} b))$	Every a must be followed by b, without any other a in
Chain Response	$\boxed{a}$•≡▶$\boxed{b}$	$\Box(a \supset \bigcirc b)$	If a is executed then b must be executed next
Chain Precedence	$\boxed{a}$≡▶•$\boxed{b}$	$\Box(\bigcirc b \supset a)$	Task b can be executed only immediately after a
Not Coexistence	$\boxed{a}$•‖•$\boxed{b}$	$\neg(\Diamond a \wedge \Diamond b)$	Only one among tasks a and b can be executed
Neg. Succession	$\boxed{a}$•‖▶•$\boxed{b}$	$\Box(a \supset \neg\Diamond b)$	Task a cannot be followed by b, and b cannot be preceded by a
Neg. Chain Succ.	$\boxed{a}$•≢▶•$\boxed{b}$	$\Box(a \supset \bigcirc\neg b)$	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

Formal Methods

- Rigorous guarantees about the behavior of computational systems

- Wide-spread industrial adoption

- Main tasks
  - **Formalisms for specs of dynamic properties:**
    - E.g., Linear Temporal Logic
  - **Verification:**
    - Check if the system satisfies specs.
  - **Synthesis:**
    - Synthesize a system that satisfies specs.

Model Checking vs Synthesis

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 ☺

Focus on Finite Traces is Shared by AI

- *In **Formal Methods** focus on **infinite traces***
- *In **AI** focus on **finite traces LTL → LTLf***

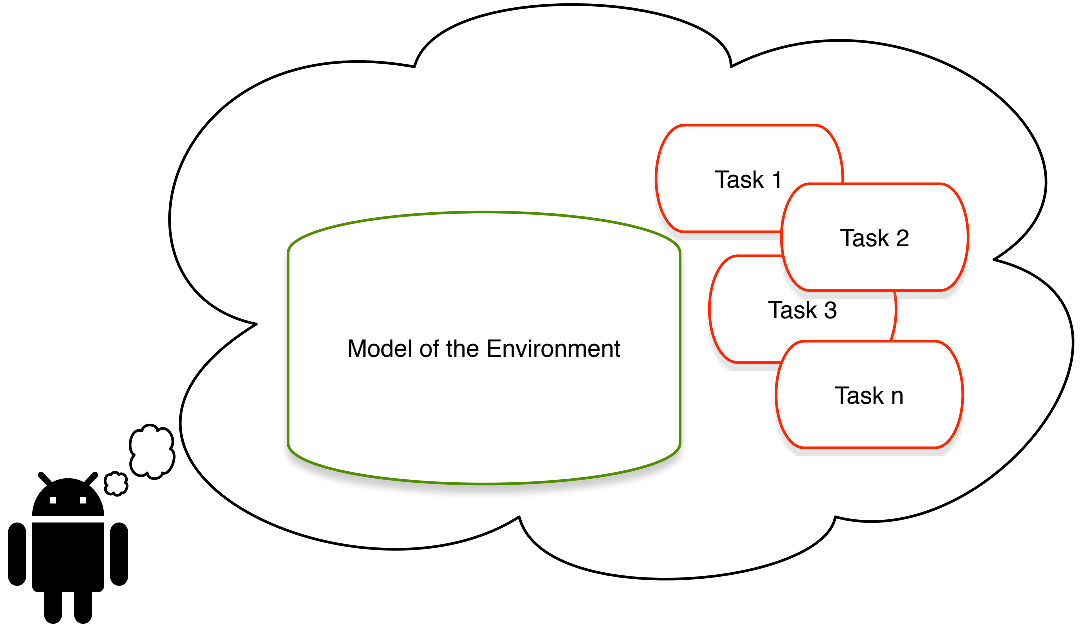
Planning in AI:

- 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



Linear Time Temporal Logics on Finite Traces

$\text{LTL}_f/\text{LDL}_f$: 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

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

Examples:	*eventually* A	"eventually A"	*reachability*
	always A	"always A"	*safety*
	always$(A \rightarrow$ *eventually* $B)$	"always if A then eventually B"	*reactiveness*
	A *until* B	"A until B"	*until*
	$\neg B$ *until* $A \vee$ *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** $\delta \doteq (\phi?;\delta)^*;\neg\phi?$

Linear Time Temporal Logics on Finite Traces

Example

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

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

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

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

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

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

- "*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*":

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

LTLf to Automata

DFA are indeed machines and hence processes!

Key point

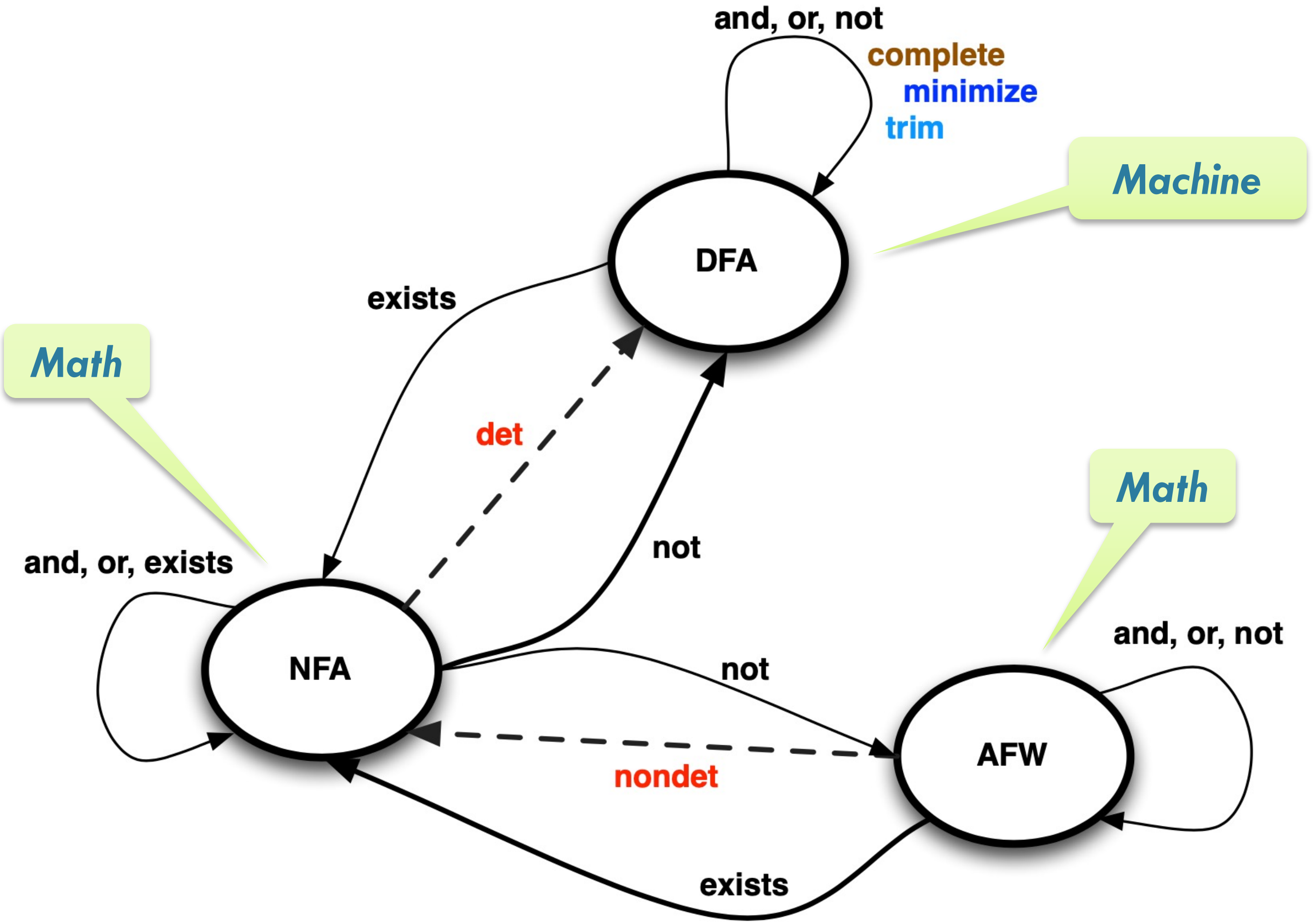
$\text{LTL}_f/\text{LDL}_f$ formulas can be translated into a finite-state automaton on finite words $\mathcal{A}_\varphi$ such that:

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

- 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



LTLf to Automata

Key point

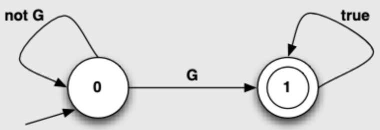
$\text{LTL}_f/\text{LDL}_f$ formulas can be translated into deterministic finite state automata (DFA).

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

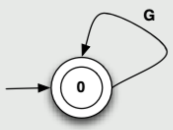
where A_φ is the DFA φ is translated into.

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

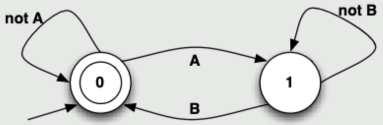
$\Diamond G$



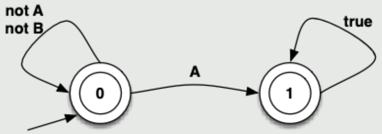
$\Box G$



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

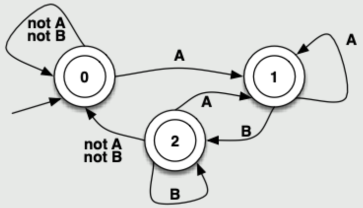


$\neg B \,\mathcal{U}\, A \vee \Box\neg B$ *"A before B"*



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

"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)

Pure Past LTL

PPLTL is a variant of LTLf that looks at traces backward (from now toward the past)

PPLTL: Pure Past LTL

$$\varphi ::= A \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \ominus\varphi \mid \varphi_1\, \mathcal{S}\, \varphi_2$$

- PPLTL has the same expressive power of LTLf.
- The DFA $\mathcal{A}_\Phi$ of a PPLTL formula Φ is worst-case single exponential (vs double-exponential for LTLf):
  - build AFA reading the trace backward (same AFA as for LTLf - linear)
  - compute the DFA of the reverse language (single exponential)
- Best algorithm for systematic translation between LTLf and PPLTL is 3EXPTIME, . . .
- . . . but in several significant cases is polynomial, e.g., all DECLARE patterns [GeattiMontaliRivkinArXiv2022].

Pure Past LTL

Actually PPLTL formulas can be evaluated in a dynamic programming method using only two instants:

- Now
- One step in the past (prev)

Current and Previous Instants

PPLTL formulas can be evaluated considering only the current and previous instants:

$$
\begin{array}{rcl}
eval(A, \Pi_{now}, \Pi_{prev}) & = & A \text{ if variable ``}A\text{'' is true in } \Pi_{now} \\
eval(\ominus\varphi, \Pi_{now}, \Pi_{prev}) & = & \ominus\varphi \text{ if variable ``}\ominus\varphi\text{'' is true in } \Pi_{prev} \\
eval(\neg\varphi, \Pi_{now}, \Pi_{prev}) & = & \neg eval(\varphi, \Pi_{now}, \Pi_{prev}) \\
eval(\varphi_1 \wedge \varphi_2, \Pi_{now}, \Pi_{prev}) & = & eval(\varphi_1, \Pi_{now}, \Pi_{prev})) \wedge eval(\varphi_2, \Pi_{now}, \Pi_{prev}) \\
eval(\varphi_1 \vee \varphi_2, \Pi_{now}, \Pi_{prev}) & = & eval(\varphi_1, \Pi_{now}, \Pi_{prev})) \vee eval(\varphi_2, \Pi_{now}, \Pi_{prev}) \\
eval(\varphi_1 \mathcal{S} \varphi_2, \Pi_{now}, \Pi_{prev}) & = & eval([\varphi_2 \vee (\varphi_1 \wedge \ominus(\varphi_1 \mathcal{S} \varphi_2))], \Pi_{now}, \Pi_{prev})
\end{array}
$$

Remember that the fixpoint equation for $\varphi_1 \mathcal{S} \varphi_2$ is
$\varphi_1 \mathcal{S} \varphi_2 \equiv \varphi_2 \vee (\varphi_1 \wedge \ominus(\varphi_1 \mathcal{S} \varphi_2))$.

Pure Past LTL

As a result one can built a symbolic DFA in linear time!

Symbolic DFA

$$\mathcal{A}_\Phi = (AP, PP, \Pi^0_{prev}, Trans, Final)$$

where

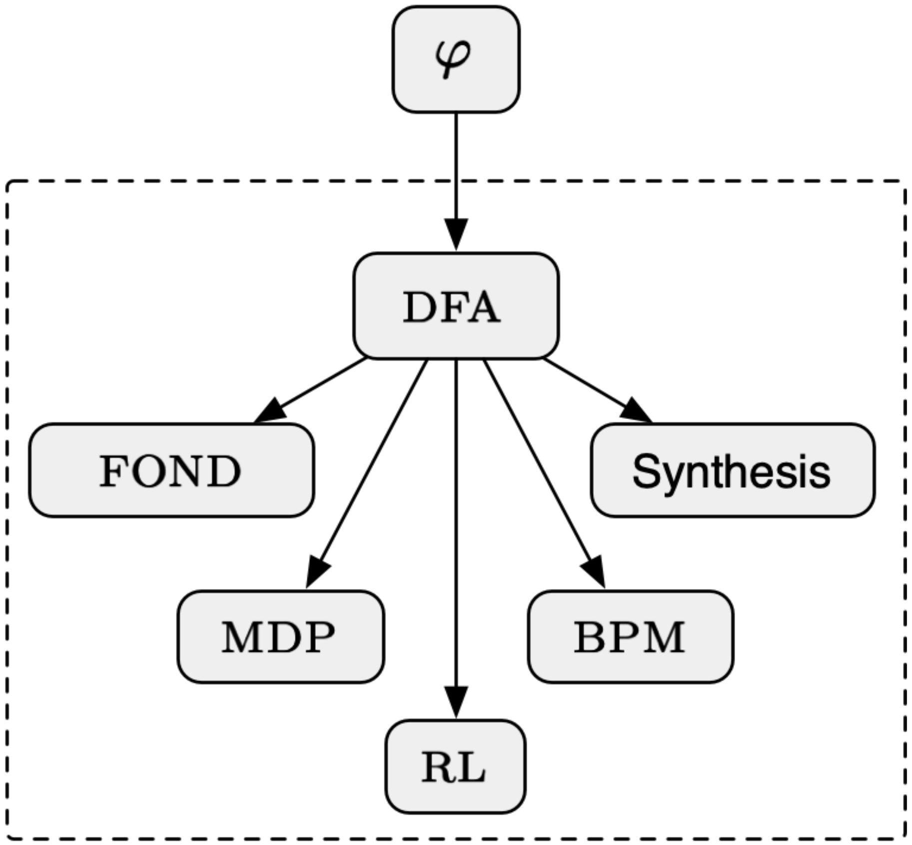
- AP set of boolean variable, one for each atomic proposition, representing which propositions are true/false in the current instant;
- PP set of boolean variable, one component for each sub-formulas the form $\ominus\varphi$ in Fisher-Ladner closure of Φ, representing which formulas were true/false at the previous instant;
- $\Pi^0_{prev} = (false, \ldots, false)$, initially all formulas of the form $\ominus\varphi$ are false
- $Trans(\Pi_{now}, \Pi_{prev}) = Trans_i(\Pi_{now}, \Pi_{prev}) \times \cdots \times Trans_n(\Pi_{now}, \Pi_{prev})$ with $n = |\Pi_{prev}|$,lat where for each variable"$\ominus\varphi_i$":
$$Trans_i(\Pi_{now}, \Pi_{prev}) = eval(\varphi_i, \Pi_{now}, \Pi_{prev})$$
- $Final(\Pi_{now}, \Pi_{prev}) \equiv eval(\Phi, \Pi_{now}, \Pi_{prev})$

Important: $\mathcal{A}_\Phi$ is linear in Φ.

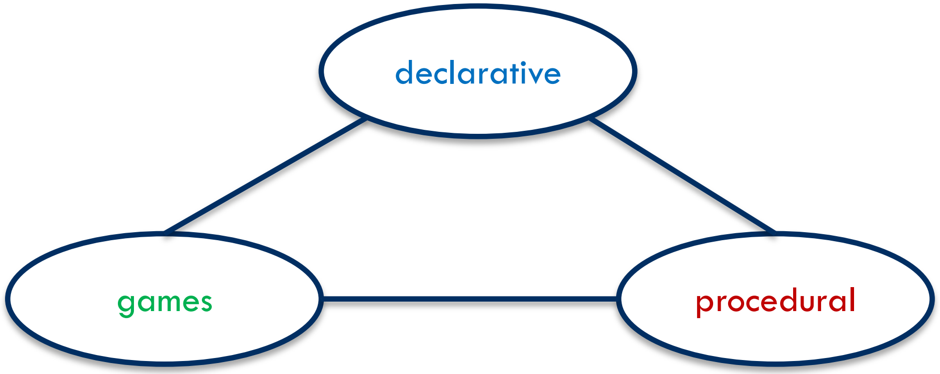
Several Applications of LTLf/PPLTL Specs

Many Applications:

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



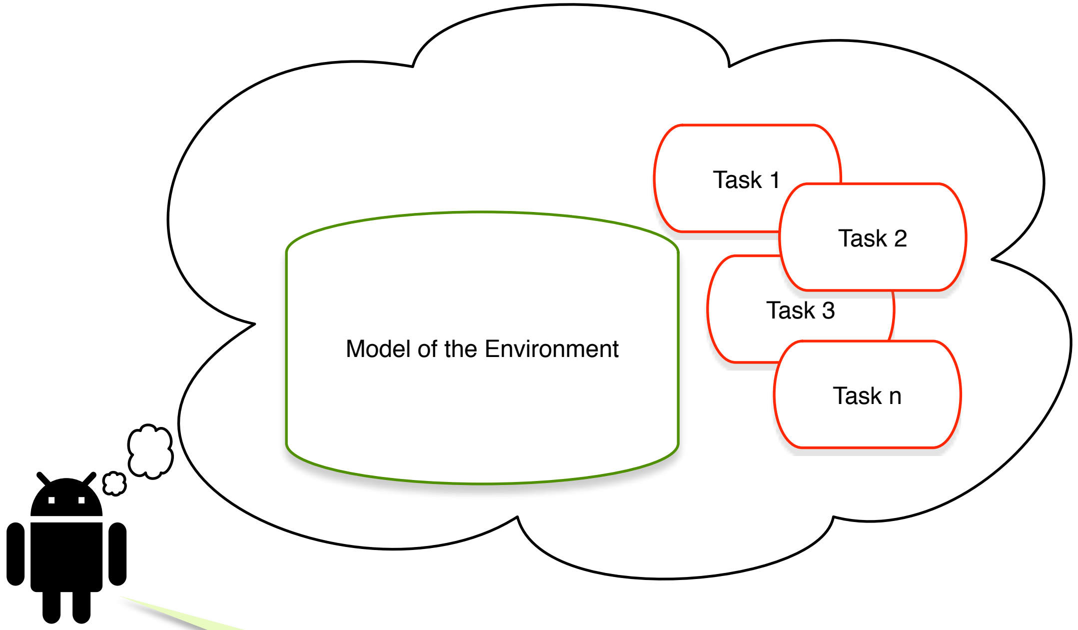
DECLARATIVE …
… TO PROCEDURAL …
… TO GAMES



Specify Models and Tasks as Processes 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** thatis **winning against ENV***

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:

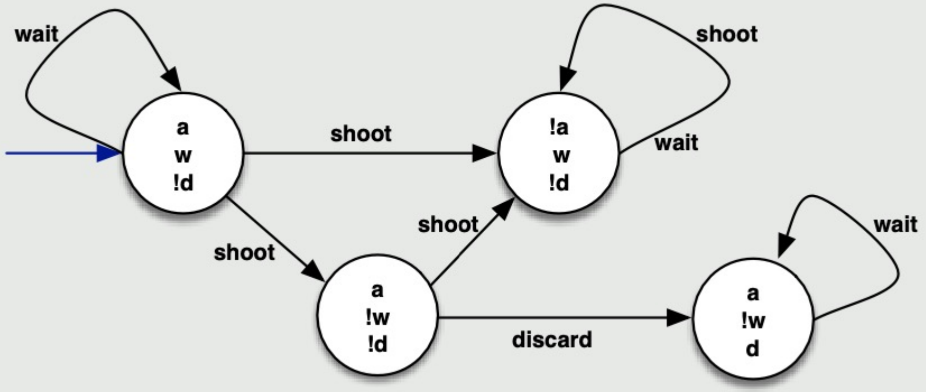
- $2^{\mathcal{F} \cup \mathcal{A}}$ alphabet (actions $\mathcal{A}$ include dummy $start$ action)
- $2^{\mathcal{F}} \cup \{s_{init}\}$ set of states
- s_{init} dummy initial state
- $F = 2^{\mathcal{F}}$ (all states of the domain are final)
- $\rho(s, [a, s']) = s'$ **with** $a \in \alpha(s)$**, and**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

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

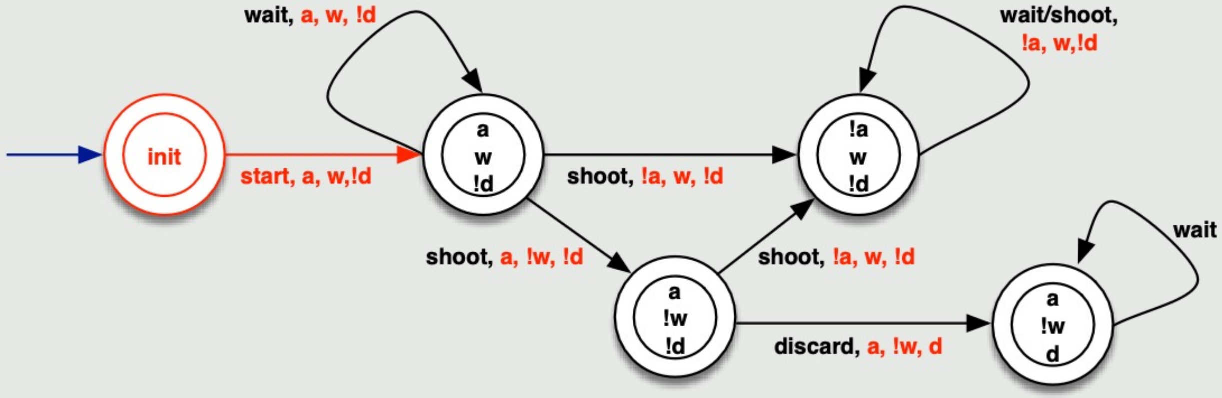
Nondeterministic Domains as Deterministic Automata

Example (Simplified Yale shooting domain)

- Domain $\mathcal{D}$:



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



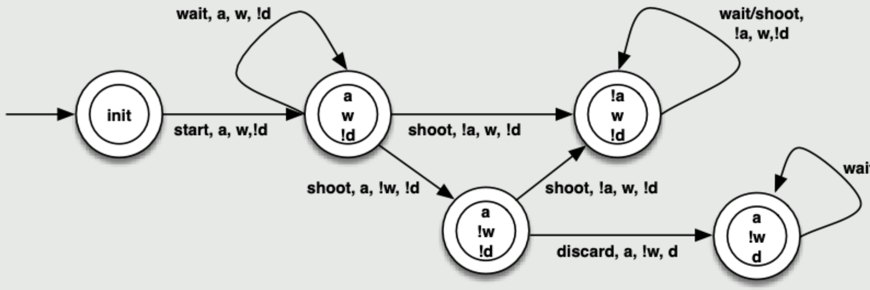
Planning in Nondeterministic Domains for Reachability Goals

Planning in nondeterministic domains

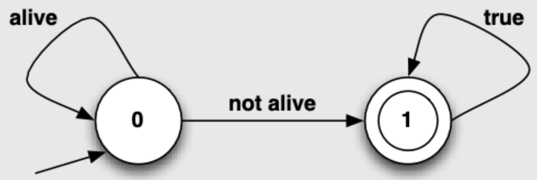
- Set the **arena** formed by all traces that satisfy both the DFA $A_{\mathcal{D}}$ for $\mathcal{D}$ and the DFA for $\Diamond G$ where G is the goal.
- Compute a **winning strategy**. *(EXPTIME-complete in $\mathcal{D}$, constant in G)*

Example (Simplified Yale shooting domain)

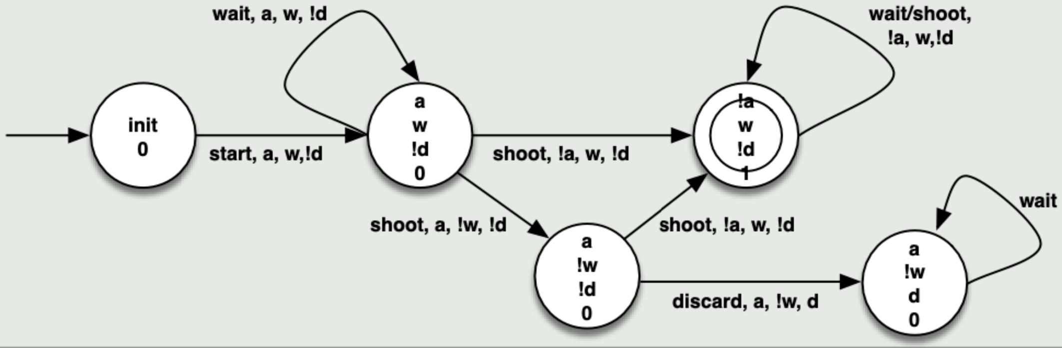
$A_{\mathcal{D}}$



$A_{\Diamond\neg a}$



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



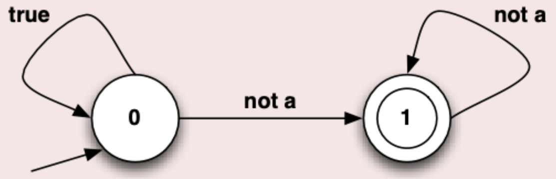
strategy

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

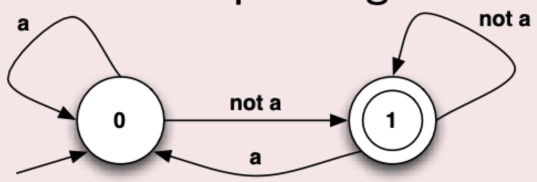
Planning in Nondeterministic Domains for Arbitrary LTLf Goals

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

NFA for $\Diamond\Box\neg a$



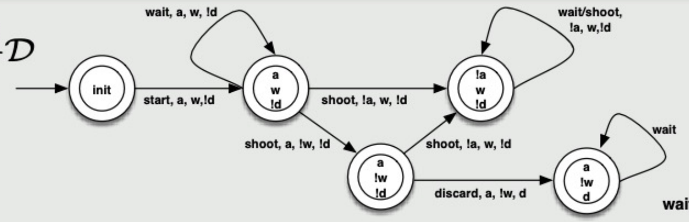
corresponding DFA



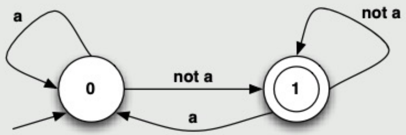
(DFA can be exponential in NFA in general)

Example (Simplified Yale shooting domain)

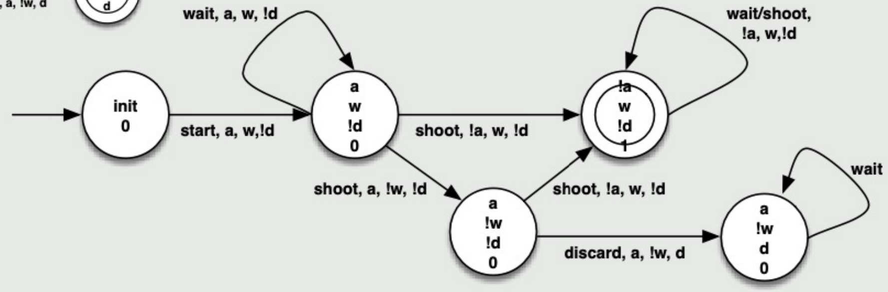
$A_{\mathcal{D}}$



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



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



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

*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***

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;
- $\varrho : S \times 2^{\mathcal{F} \cup \mathcal{A}} \to S$, 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.

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 $f : (2^{\mathcal{F}})^* \to \mathcal{A}$ that, given a **history of choices from the environment**, decides which action $\mathcal{A}$ to do next.
- A **winning strategy** is a strategy $f : (2^{\mathcal{F}})^* \to \mathcal{A}$ 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:

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

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

It's a game agent vs env!

- Build the arena
- Play to win!

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

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)

Planning in Nondeterministic Domains for PPLTL Goals

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

FOND for LTL_f goals

Algorithm: FOND for LTL_f/ PPLTL 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) single exp for PPLTL
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);* Same as classic FOND
- *2EXPTIME-complete in the goal.* EXPTIME-complete for PPLTL

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

Planning in Nondeterministic Domains for *PPLTL* Goals

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

FOND for LTL_f goals

1. Given a FOND domain D and an PPLTL formula φ
2. Compute the symbolic DFA A_φ for φ (linear)
   - *using quoted subformulas as fluent*
3. Encode the symbolic DFA A_φ in FOND (linear)
   - *requires conditional effects*
4. Plan for the propositional goal "φ" (exponential)
5. Return (strong/strong cyclic) plan.

Theorem ([DeGiacomoRubinIJCAI18])

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

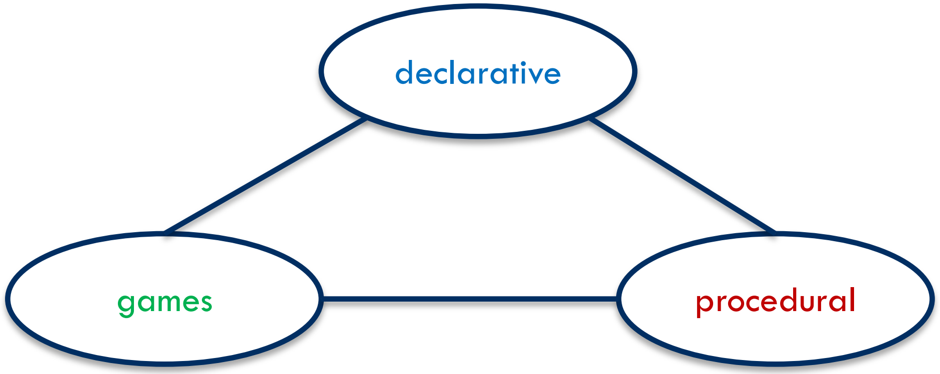
- *EXPTIME-complete in the domain (assuming a logarithmic representation as in PDDL);*
- *2EXPTIME-complete in the goal.* *EXPTIME-complete for PPLTL*

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)

DECLARATIVE …
… TO PROCEDURAL …
… TO GAMES

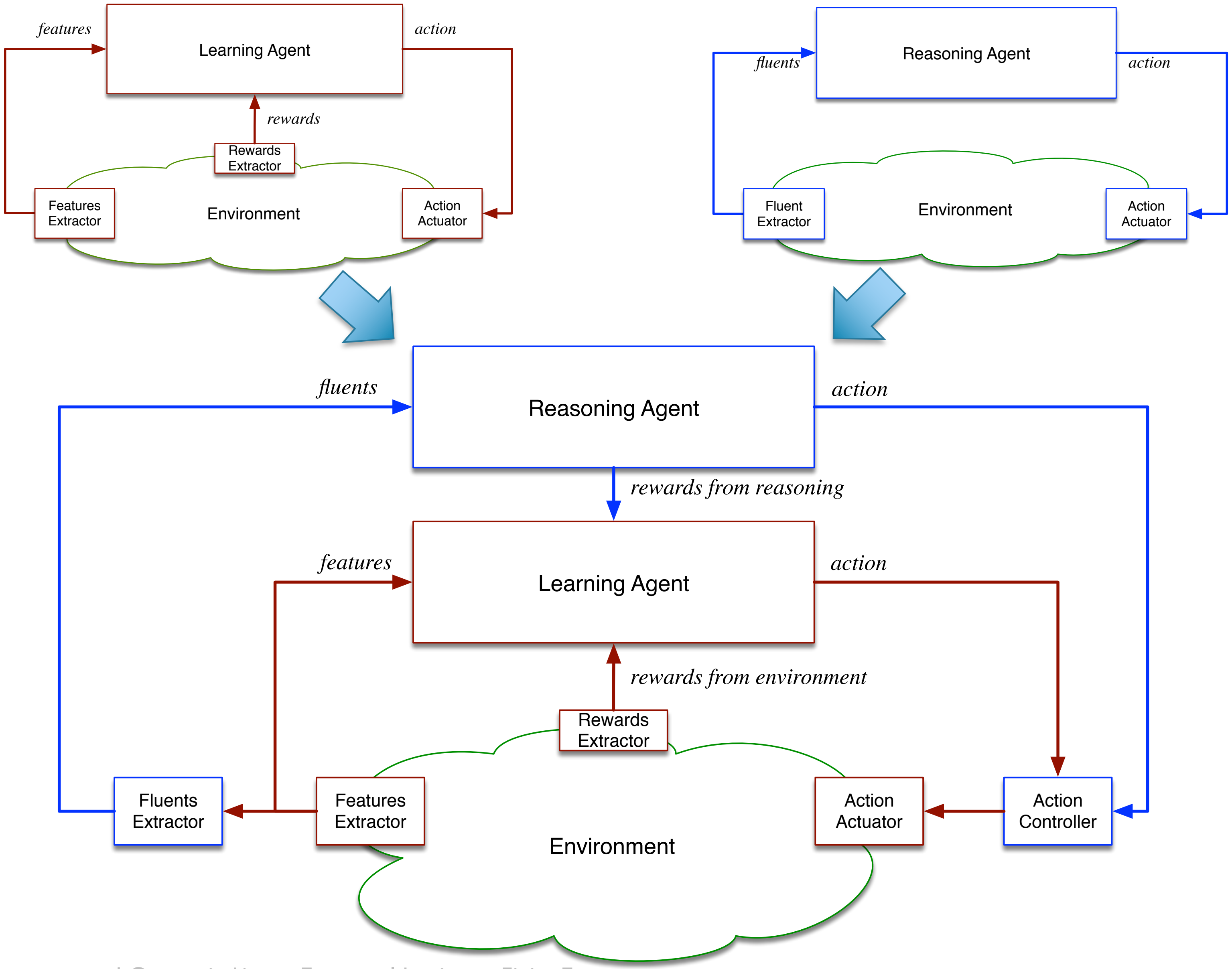
DECLARATIVE CONTROL IN RL



Combining 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



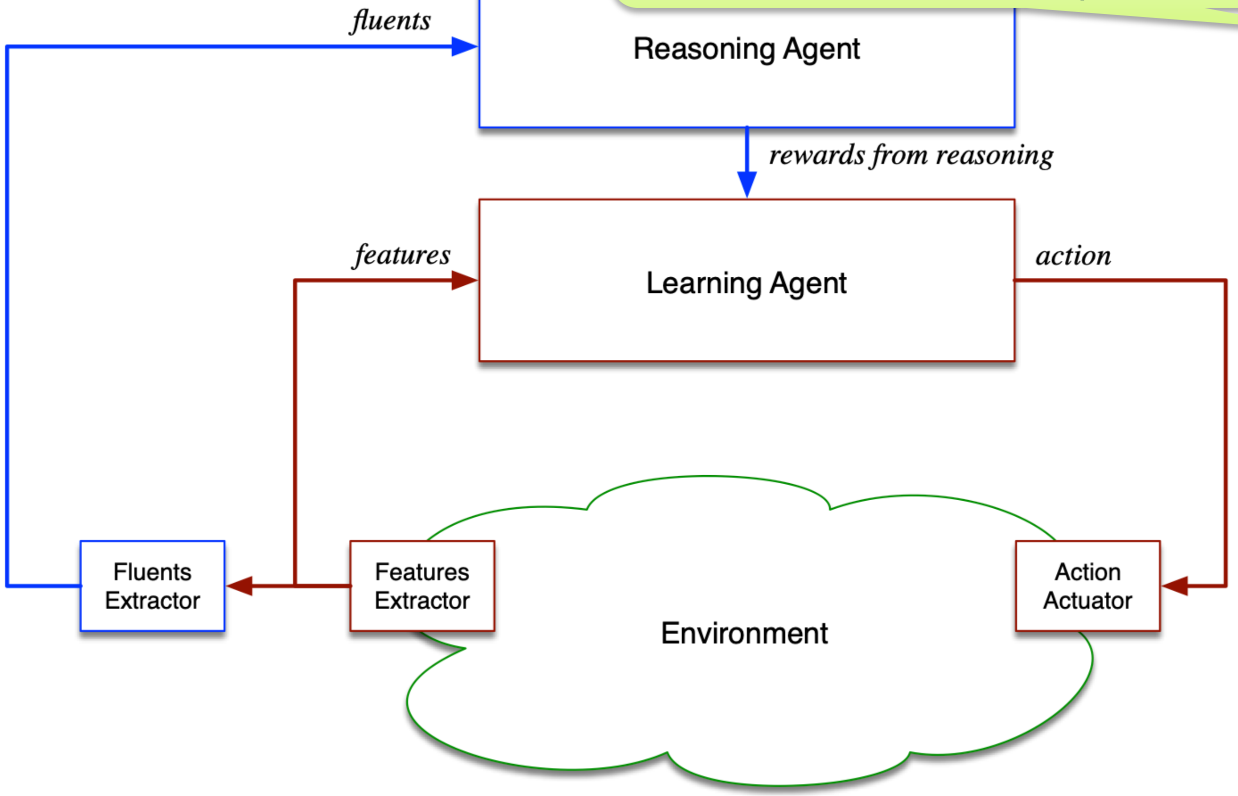
Reinforcement Learning with LTLf non-Markovian Rewards

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

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 $\Rightarrow$ $\overline{R}_{ag} : (S_{ag}, A_{ag})^* \to \mathbb{R}$
  non-Markovian rewards!
- **Mapping between S_{ag} and $\mathcal{L}$**

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



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

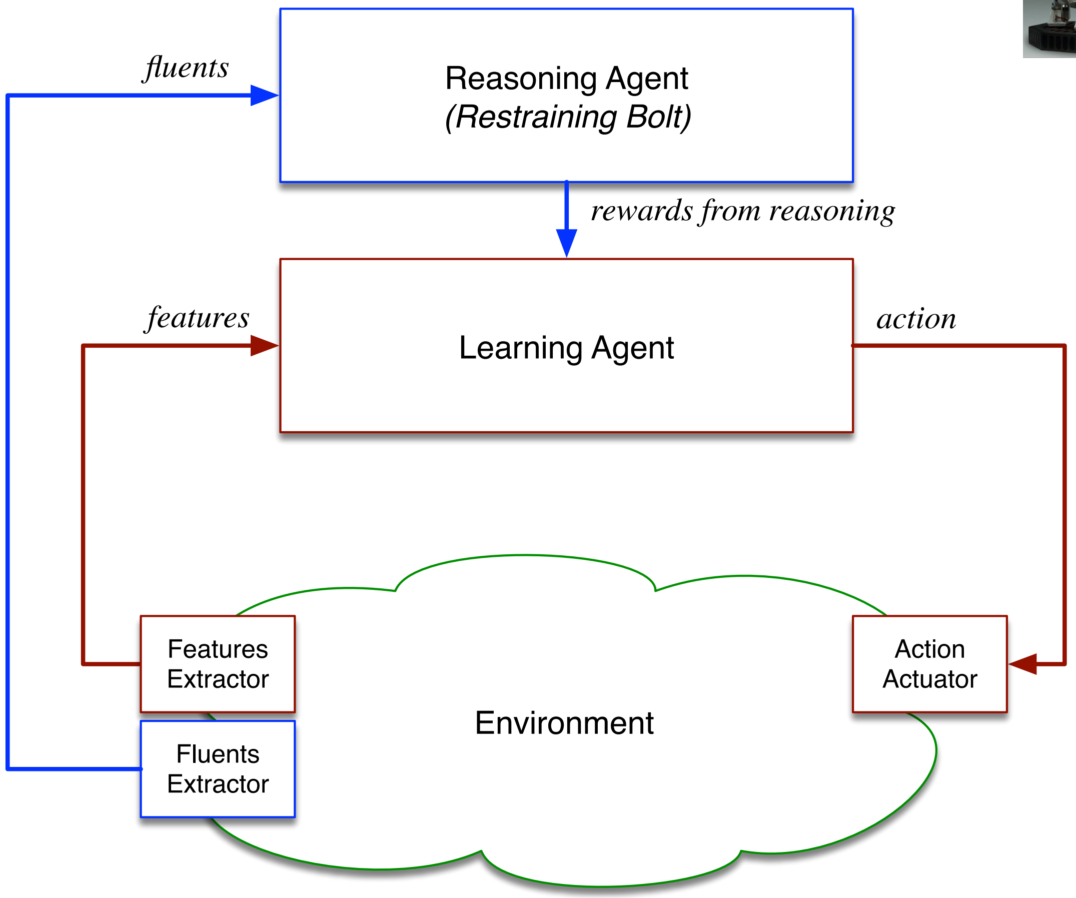
- Reasoning agent: $\mathcal{R} = (\mathcal{L}, \{(\varphi_i, r_i)\}_{i=1}^{m})$

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

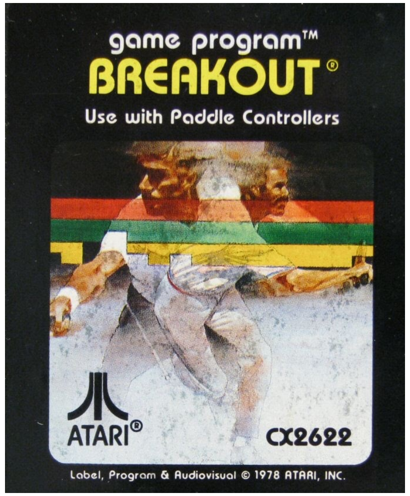
  non-Markovian rewards!

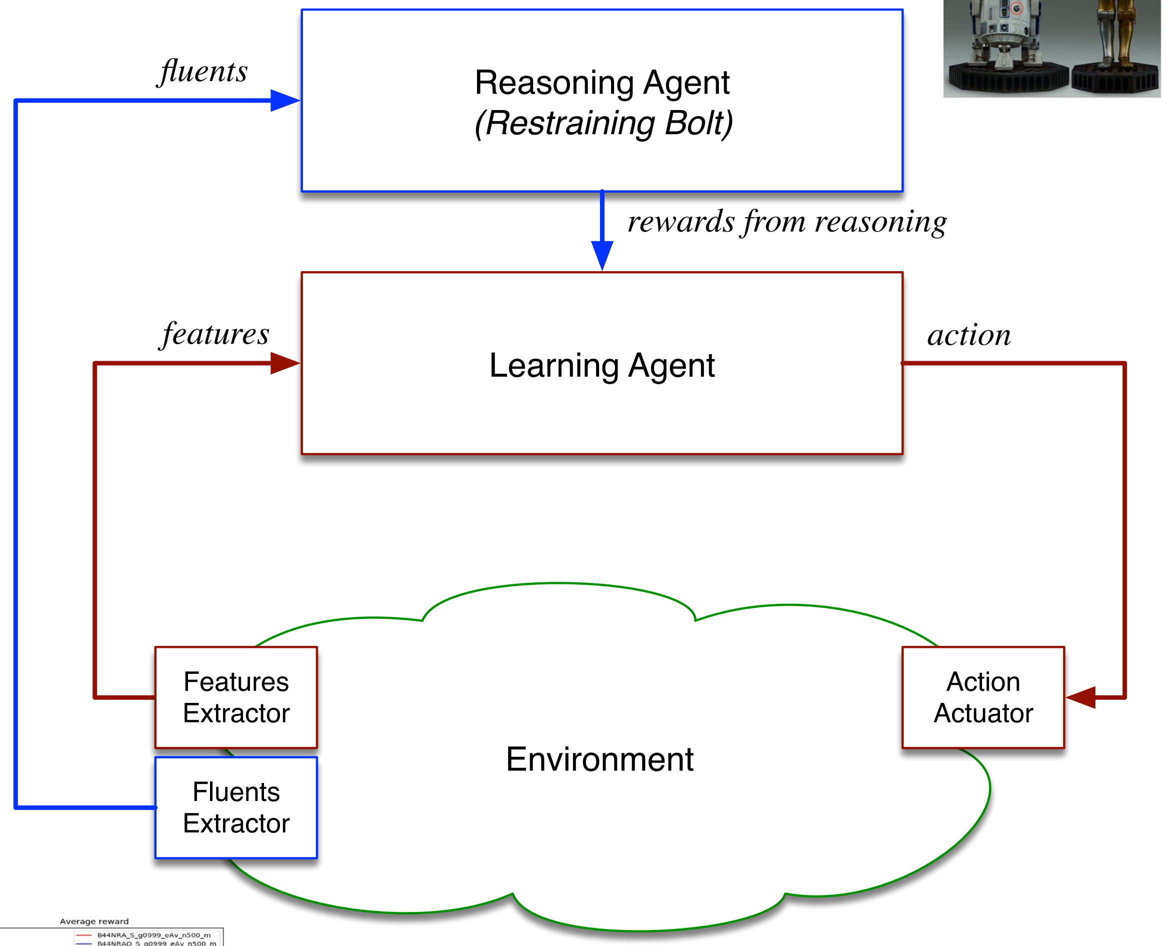
- ~~Mapping between S_{ag} and $\mathcal{L}$~~

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



Example: Breakout



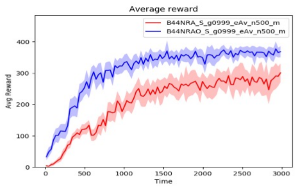


Learning Agent

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

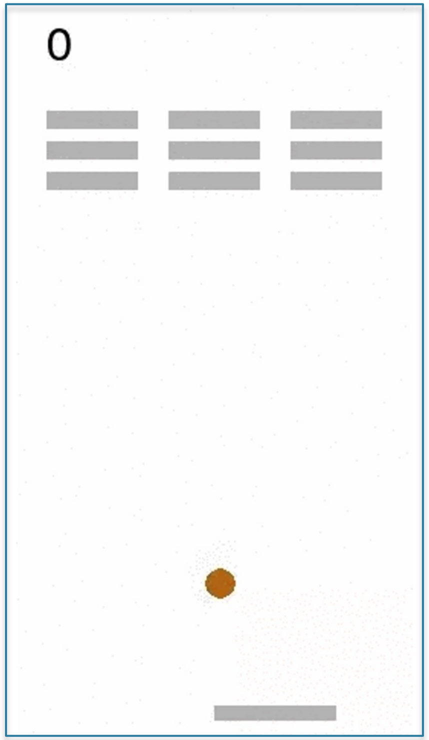
Restraining Bolt (Reasoning Agent)

- **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)

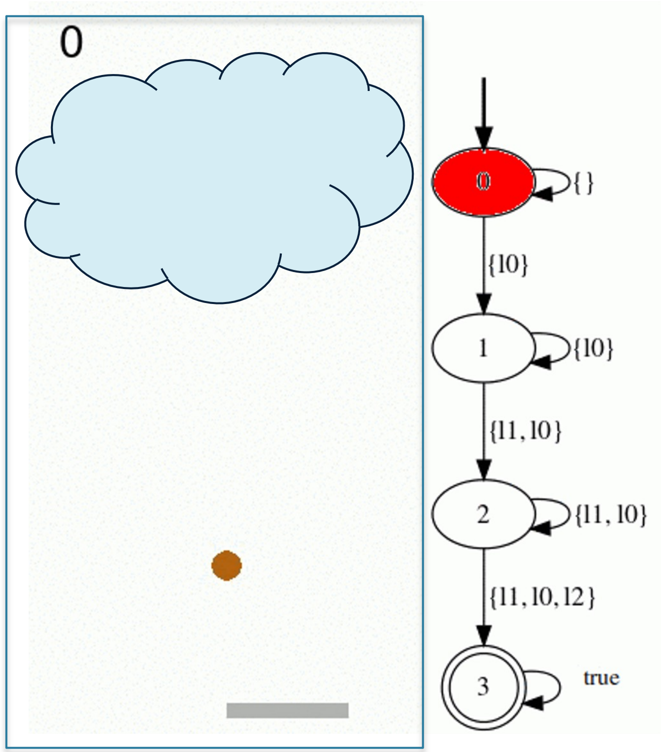


https://sites.google.com/diag.uniroma1.it/restraining-bolt

The Agent Ignores the Fluents!



How the world is

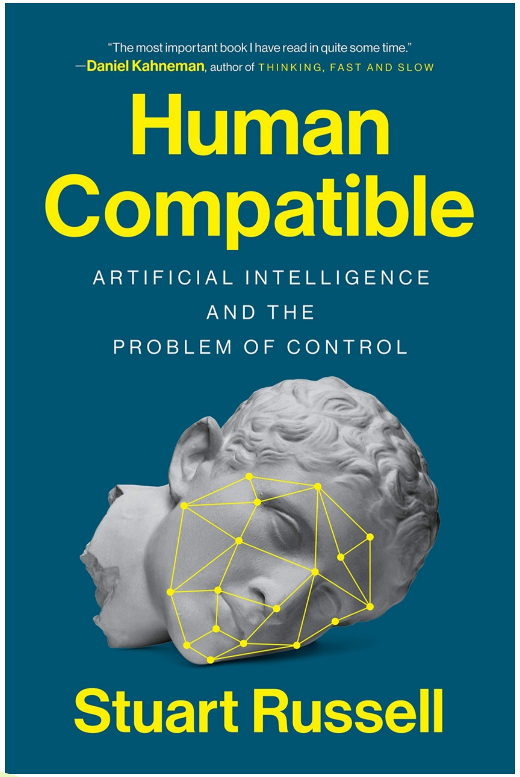


How the agent sees the world

Controlling RL with Logic for Safety

The idea of restraining bolt can be subscribed to that part of research generated by the urgency of providing **safety guarantees** to AI techniques based on learning.

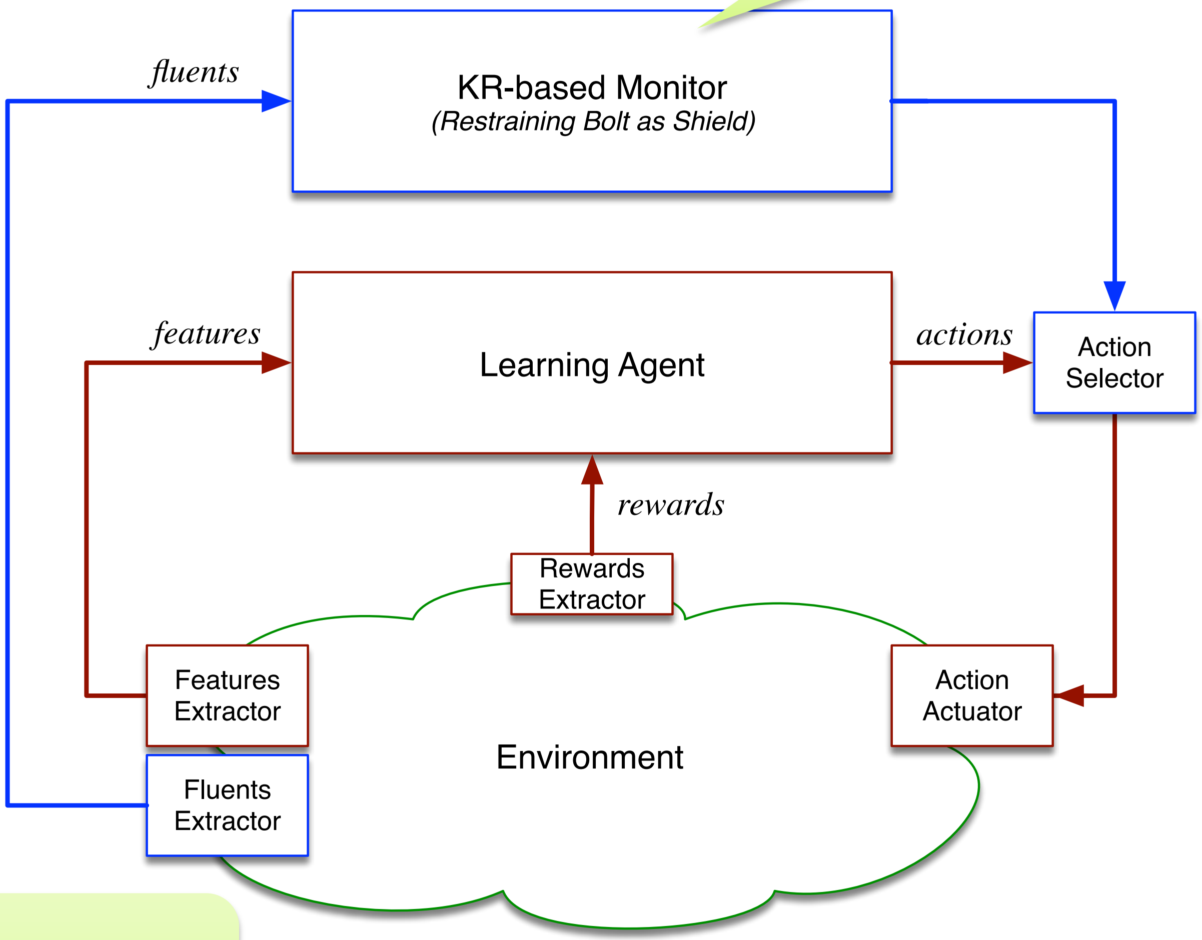
- S. Russell, D. Dewey, and M. Tegmark. **Research priorities for robust and beneficial artificial intelligence.** AI Magazine, 36(4), 2015.
- ACM U.S. Public Policy Council and ACM Europe Policy Committee. **Statement on algorithmic transparency and accountability.** ACM, 2017.
- D. Hadfield-Menell, A. D. Dragan, P. Abbeel, and S. J. Russell. **The off-switch game.** In IJCAI 2017.
- D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mane. **Concrete problems in AI safety**. CoRR, abs/1606.06565, 2016.
- Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Konighofer, Scott Niekum, Ufuk Topcu: **Safe Reinforcement Learning via Shielding**. AAAI 2018.
- Min Wen, Rüdiger Ehlers, Ufuk Topcu: **Correct-by-synthesis reinforcement learning with temporal logic constraints** IROS 2015.



Shields

Restraining Bolts as Shields for Frame

$Poss(a,h)$ iff $h \models \varphi_{LTLf}$



Separate representations

- KR-based agent
- RL-based agent

Shielded execution

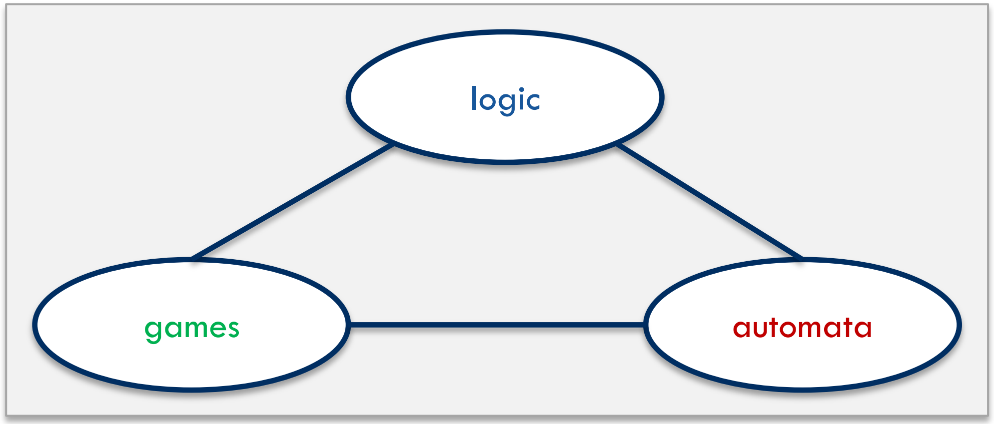
- KR-based agent acts as a monitor
- It disallows forbidden actions

Very related to "Framed Autonomy" in ABPMS, see ***AI-Augmented Business Process Management Systems: A Research Manifesto*** *ACM Transaction on Management Information Systems, 2023*

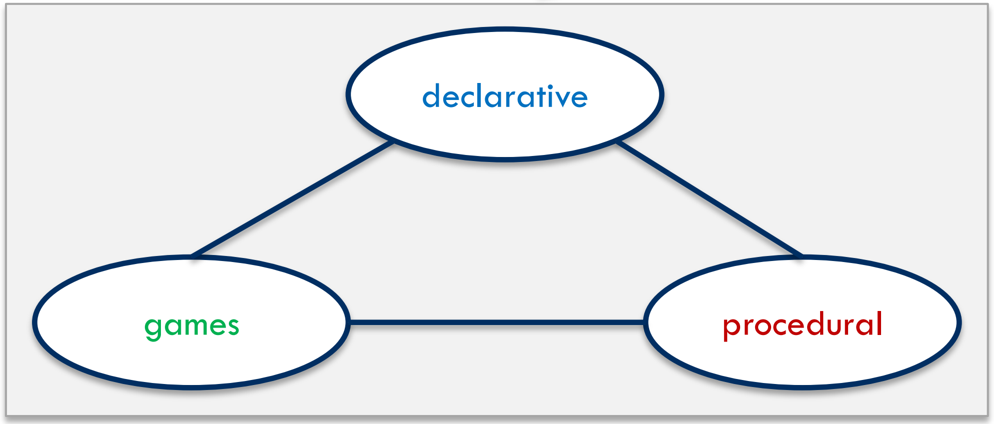
N. Alechina, M. Dastani, G. De Giacomo, B. Logan, G. Perelli, G. Varricchione. Pure-Past Action Masking, AAAI2024

Conclusion

Powerful formal framework for controller synthesis



It has profound implication in how we can represent services



The instantiation on LTLf/PPLTL is particularly effective