Linear Temporal Logics on Finite Traces

AAAI 2023 Spring Symposium
On the Effectiveness of Temporal Logics on Finite Traces in AI

Giuseppe De Giacomo

ERC Advanced Grant
WhiteMech:
White-box Self Programming Mechanisms

Outline

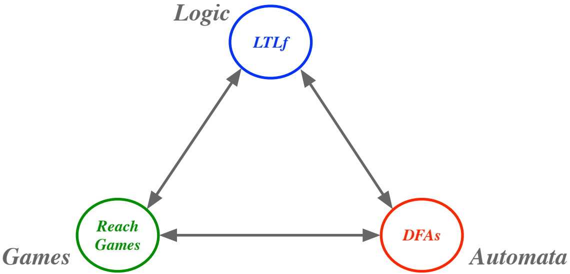
Outline

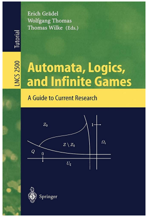
Motivation: AI

We are interested in building

AI Agents

Linear temporal logics on finite traces are a fantastic tool for this enterprise, because it gives computational concreteness to the famous **Logics-Automata-Games** triangle from Formal Methods:

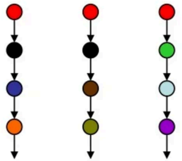




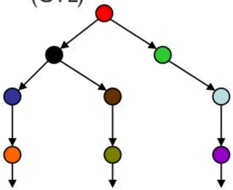
Motivation: AI

Temporal logic

- Linear Time
  - Every moment has a unique successor
  - Infinite sequences (words)
  - Linear Time Temporal Logic (LTL)



- Branching Time
  - Every moment has several successors
  - Infinite tree
  - Computation Tree Logic (CTL)



Courtesy of Carlo Gezzi

See Anneline Daggelinckx's talk!

Motivation: AI

Artificial Intelligence and in particular the Knowledge Representation and Planning community well aware of temporal logics since a long time.

Foundations borrowed from temporal logics studied in CS, in particular:
Linear Temporal Logic (LTL) [Pnueli77].

However:

- Often, LTL is interpreted on finite trajectories/traces.

- MetateM: logic programming in LTL [BarringerFisherGabbayGoughOwens89] - infinite/finite
- Temporally extended goals [BacchusKabanza96] - infinite/finite
- Temporal constraints on trajectories [GereviniHslumLongSaettiDimopoulos09 - 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]

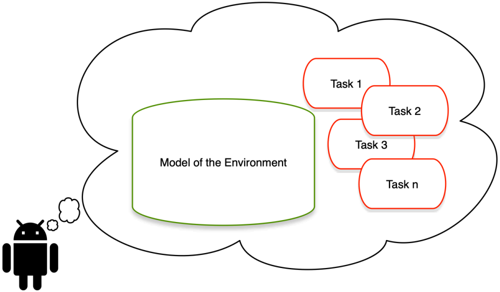
Motivation: AI

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 terminate?

- 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

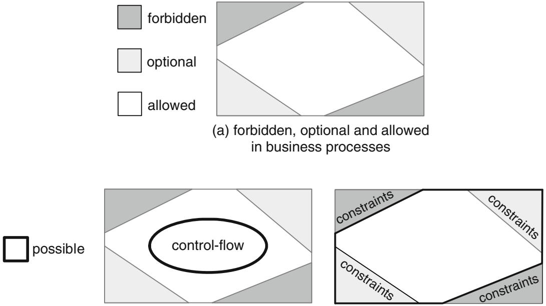


See Yves Lesperance's talk!

Motivation: BPM

Business Process Management community has proposed a declarative approach to business process modeling based on LTL on finite traces: DECLARE

Basic idea: Drop explicit representation of processes, and LTL formulas specify the allowed finite traces.
[VanDerAalstPesic06] [PesicBovsnavkiDraganVanDerAalst10].



(a) forbidden, optional and allowed in business processes

(b) procedural workflow

(c) declarative workflow

See Marco Montali's talk!

Outline

LTL over finite traces

LTL$_f$: the language (in symbols)

Same syntax as standard LTL but interpreted over finite traces

$$\varphi ::= A \mid \quad \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \supset \varphi_2 \mid \quad \bigcirc\varphi \mid \newmoon\varphi \mid \Diamond\varphi \mid \Box\varphi \mid \varphi_1 \,\mathcal{U}\, \varphi_2$$

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

LTL over finite traces

LTL_f: the language (in words)

Note: we do not need fancy symbols we can use english words instead:

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

In symbols

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

But see the paper

Ben Greenman, Sam Saarinen, Tim Nelson, Shriram Krishnamurthi. ***Little Tricky Logic: Misconceptions in the Understanding of LTL****, Art Sci. Eng. Program. 7(2), 2023*

Ben will test how LTL_f is tricky as well on us during this workshop :-)

LTL_f Examples

Example

Consider the following formula:

$$\Diamond A$$

- On infinite traces:
- On finite traces:

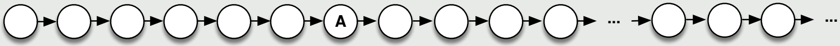
LTL_f Examples

Example

Consider the following formula:

$$\Diamond A$$

- On infinite traces:



- On finite traces:

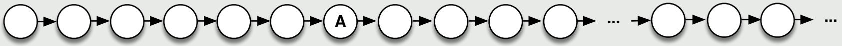
LTL_f Examples

Example

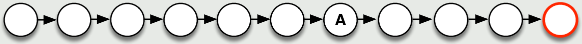
Consider the following formula:

$$\Diamond A$$

- On infinite traces:



- On finite traces:



LTL$_f$ Examples

Example

Consider the following formula:

$$\Box A$$

- On infinite traces:
- On finite traces:

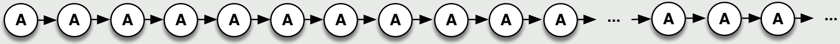
LTL_f Examples

Example

Consider the following formula:

$$\Box A$$

- On infinite traces:



- On finite traces:

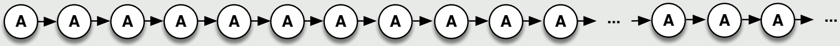
LTL_f Examples

Example

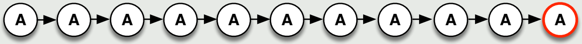
Consider the following formula:

$$\Box A$$

- On infinite traces:



- On finite traces:



Example

Consider the following formula:

$$\Diamond \bigcirc A$$

- On infinite traces:
- On finite traces:

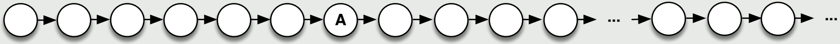
LTL_f Examples

Example

Consider the following formula:

$$\Diamond \bigcirc A$$

- On infinite traces:



- On finite traces:

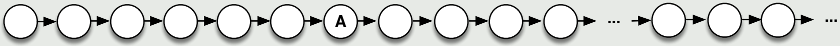
LTL$_f$ Examples

Example

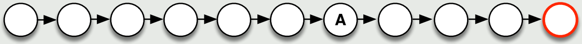
Consider the following formula:

$$\Diamond \bigcirc A$$

- On infinite traces:



- On finite traces:



LTL_f Examples

Example

Consider the following formula:

$$\Box \bigcirc A$$

- On infinite traces:
- On finite traces:

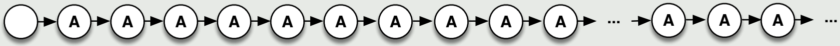
LTL$_f$ Examples

Example

Consider the following formula:

$$\Box \bigcirc A$$

- On infinite traces:



- On finite traces:

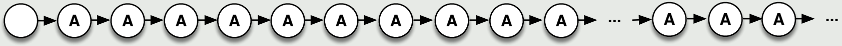
LTL$_f$ Examples

Example

Consider the following formula:

$$\Box \bigcirc A$$

- On infinite traces:



- On finite traces:

None!!!

LTL$_f$ Examples

Example

Consider the following formula:

$$\Box \bullet A$$

- On infinite traces (in LTL $\bullet A$ must be replaced by $\neg \circ \neg A$ which is equivalent to $\circ A$):
- On finite traces:

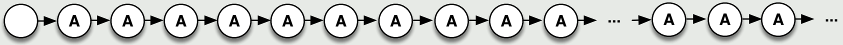
LTL_f Examples

Example

Consider the following formula:

$$\Box \bullet A$$

- On infinite traces (in LTL $\bullet A$ must be replaced by $\neg \bigcirc \neg A$ which is equivalent to $\bigcirc A$):



- On finite traces:

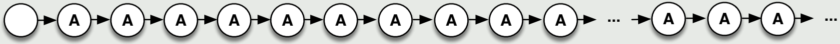
LTL$_f$ Examples

Example

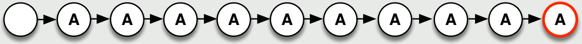
Consider the following formula:

$$\Box \bullet A$$

- On infinite traces (in LTL $\bullet A$ must be replaced by $\neg \circ \neg A$ which is equivalent to $\circ A$):



- On finite traces:



Capturing STRIPS

Example (Capturing STRIPS Planning as $\textsc{LTL}_f$ SAT)

- For each action $A \in Act$ with precondition φ and effects $\bigwedge_{F \in Add(A)} F \wedge \bigwedge_{F \in Del(A)} \neg F$
  - $\Box(\bigcirc A \supset \varphi)$: if next action A has occurred (denoted by a proposition A) then now precondition φ must be true;
  - $\Box(\bigcirc A \supset \bigcirc(\bigwedge_{F \in Add(A)} F \wedge \bigwedge_{F \in Del(A)} \neg F))$: when A occurs, its effects are true;
  - $\Box(\bigcirc A \supset \bigwedge_{F \notin Add(A) \cup Del(A)} (F \equiv \bigcirc F))$: everything not in add or delete list, remains unchanged.
- At every step one and only one action is executed: $\Box((\bigvee_{A \in Act} A) \wedge (\bigwedge_{A_i, A_j \in Act, A_i \neq A_j} A_i \supset \neg A_j))$.
- Initial situation is described as the conjunction of propositions $Init$ that are true/false at the beginning of the trace: $\bigwedge_{F \in Init} F \wedge \bigwedge_{F \notin Init} \neg F$.
- Finally goal φ_g eventually holds: $\Diamond \varphi_g$.

Thm: A plan exists iff the $\textsc{LTL}_f$ *formula is SAT.*

Capturing SitCalc

Example (Propositional SitCalc Basic Action Theories in LTL_f)

- Successor state axiom (instantiated for each action A) $F(do(A, s)) \equiv \varphi^+(s) \vee (F(s) \wedge \neg\varphi^-(s))$ can be fully captured:

$$\Box(\bigcirc A \supset (\bigcirc F \equiv \varphi^+ \vee F \wedge \neg\varphi^-).$$

- Precondition axioms $Poss(A, s) \equiv \varphi_A(s)$ can only be captured in the part saying "if A happens then its precondition must be true":

$$\Box(\bigcirc A \supset \varphi_A).$$

The part saying "if the precondition φ_A holds then action A is ***possible****" cannot be expressed in linear time formalisms, since they talk about traces that actually happen not the ones that are possible.*

Examples from DECLARE

name of template	*LTL semantics*
$\textit{responded existence}(A, B)$	$\Diamond A \Rightarrow \Diamond B$
$\textit{co-existence}(A, B)$	$\Diamond A \Leftrightarrow \Diamond B$
$\textit{response}(A, B)$	$\Box(A \Rightarrow \Diamond B)$
$\textit{precedence}(A, B)$	$(\neg B\ \mathcal{U} A) \vee \Box(\neg B)$
$\textit{succession}(A, B)$	$\textit{response}(A, B) \wedge \textit{precedence}(A, B)$
$\textit{alternate response}(A, B)$	$\Box(A \Rightarrow \bigcirc(\neg A\ \mathcal{U} B))$
$\textit{alternate precedence}(A, B)$	$\textit{precedence}(A, B) \wedge \Box(B \Rightarrow \bigcirc(\textit{precedence}(A, B)))$
$\textit{alternate succession}(A, B)$	$\textit{alternate response}(A, B) \wedge \textit{alternate precedence}(A, B)$
$\textit{chain response}(A, B)$	$\Box(A \Rightarrow \bigcirc B)$
$\textit{chain precedence}(A, B)$	$\Box(\bigcirc B \Rightarrow A)$
$\textit{chain succession}(A, B)$	$\Box(A \Leftrightarrow \bigcirc B)$

name of template	*LTL semantics*
$\textit{not co-existence}(A, B)$	$\neg(\Diamond A \wedge \Diamond B)$
$\textit{not succession}(A, B)$	$\Box(A \Rightarrow \neg(\Diamond B))$
$\textit{not chain succession}(A, B)$	$\Box(A \Rightarrow \bigcirc(\neg B))$

name of template	*LTL semantics*
$\textit{existence}(1, A)$ $\textit{existence}(2, A)$ ... $\textit{existence}(n, A)$	$\Diamond A$ $\Diamond(A \wedge \bigcirc(\textit{existence}(1, A)))$ ... $\Diamond(A \wedge \bigcirc(\textit{existence}(n-1, A)))$
$\textit{absence}(A)$	$\neg\textit{existence}(1, A)$
$\textit{absence}(2, A)$ $\textit{absence}(3, A)$ ... $\textit{absence}(n+1, A)$	$\neg\textit{existence}(2, A)$ $\neg\textit{existence}(3, A)$ ... $\neg\textit{existence}(n+1, A)$
$\textit{init}(A)$	A

See Luca Geatti's talk!

Weak Until and Release in LTL_f

Weak Until

Weak Until, denoted by $\varphi \mathcal{W} \psi$ says that "φ holds until ψ holds, however it is fine for ψ not to hold at all, and in that case φ holds forever". Note this is the typical interpretation of the word "until" in English. Formally it is defined as:

$$\varphi_1 \mathcal{W} \varphi_2 \doteq (\varphi_1 \,\mathcal{U}\, \varphi_2) \vee \Box \varphi_1$$

Release

Release denoted by $\varphi \mathcal{R} \psi$ says that "φ releases ψ from holding forever". It can be defined as:

$$\varphi_1 \mathcal{R} \varphi_2 \doteq \varphi_1 \mathcal{W} (\varphi_1 \wedge \varphi_2)$$

The following holds:

- $\varphi_1 \mathcal{R} \varphi_1 \equiv \neg(\neg \varphi_1 \,\mathcal{U}\, \neg \varphi_2)$
- it also holds that $\varphi_1 \mathcal{U} \varphi_2 \equiv \neg(\neg \varphi_1 \mathcal{R} \neg \varphi_2)$

(Release is dual of Until)

Weak Until and Release in LTL_f

Weak Until

Weak Until, denoted by $\varphi \mathcal{W} \psi$ says that "φ holds until ψ holds, however it is fine for ψ not to hold at all, and in that case φ holds forever". Note this is the typical interpretation of the word "until" in English. Formally it is defined as:

$$\varphi_1 \mathcal{W} \varphi_2 \doteq (\varphi_1 \,\mathcal{U}\, \varphi_2) \vee \Box \varphi_1$$

Release

Release denoted by $\varphi \mathcal{R} \psi$ says that "φ releases ψ from holding forever". It can be defined as:

$$\varphi_1 \mathcal{R} \varphi_2 \doteq \varphi_1 \mathcal{W} (\varphi_1 \wedge \varphi_2)$$

The following holds:

- $\varphi_1 \mathcal{R} \varphi_1 \equiv \neg(\neg\varphi_1 \,\mathcal{U}\, \neg\varphi_2)$
- it also holds that
  $\varphi_1 \mathcal{U} \varphi_2 \equiv \neg(\neg\varphi_1 \mathcal{R} \neg\varphi_2)$

(Release is dual of Until)

Fixpoint Equivalences in LTL_f

Introduced since the early days of LTL in CS, for connection with fixpoint theory and tableaux expansion rules, [GabbayPnueliShelahStavi80],[Manna82],[Emerson90]

- $\Diamond\varphi \equiv \varphi \vee \bigcirc(\Diamond\varphi)$ –then choose lfp
- $\Box\varphi \equiv \varphi \wedge \bullet(\Box\varphi)$ –then choose gfp *(Note: in LTL_f, differently from LTL, $\Box\varphi \equiv \varphi \wedge \bigcirc(\Box\varphi)$ does* ***not*** *hold.)*
- $\varphi_1 \,\mathcal{U}\, \varphi_2 \equiv \varphi_2 \vee (\varphi_1 \wedge \bigcirc(\varphi_1 \,\mathcal{U}\, \varphi_2))$ –then choose lfp
- $\varphi_1 \mathcal{R} \varphi_2 \equiv \varphi_2 \wedge (\varphi_1 \vee \bullet(\varphi_1 \mathcal{R} \varphi_2))$ –then choose gfp

Fixpoint Equivalences in LTL$_f$ and "next normal form"

By recursively applying fixpoint equivalences, considering as base case propositions and formulas prefixed with $\bigcirc$ or $\bullet$, i.e.:

$$
\begin{array}{rcl}
nextNF(A) & = & A \\
nextNF(\bigcirc\varphi) & = & \bigcirc\varphi \\
nextNF(\bullet\varphi) & = & \bullet\varphi \\
nextNF(\neg\varphi) & = & \neg nextNF(\varphi) \\
nextNF(\varphi_1 \wedge \varphi_2) & = & nextNF(\varphi_)\wedge nextNF(\varphi_2) \\
nextNF(\varphi_1 \vee \varphi_2) & = & nextNF(\varphi_)\vee nextNF(\varphi_2)
\end{array}
\qquad
\begin{array}{rcl}
nextNF(\Diamond\varphi & = & nextNF(\varphi) \vee \bigcirc(\Diamond\varphi) \\
nextNF(\Box\varphi) & = & nextNF(\varphi) \wedge \bullet(\Box\varphi) \\
nextNF(\varphi_1\,\mathcal{U}\,\varphi_2) & = & nextNF(\varphi_2) \vee (nextNF(\varphi_1) \wedge \bigcirc(\varphi_1\,\mathcal{U}\,\varphi_2)) \\
nextNF(\varphi_1\mathcal{R}\varphi_2) & = & nextNF(\varphi_2) \wedge (nextNF(\varphi_1) \vee \bullet(\varphi_1\mathcal{R}\varphi_2))
\end{array}
$$

we get that every formula φ in LTL$_f$ (or LTL, LDL$_f$, Pure Past LTL) can be decomposed is equivalent to a formula of the form

$$\varphi \equiv Bool(A, \bigcirc\varphi, \bullet\varphi)$$

that is φ gets partitioned into a part that to be evaluated NOW and a part that to be evaluated NEXT.

This observation is at the base of many results, including, e.g.:

- translation of LTL into alternating automata [Vardi95]
- Bacchus&Kabanza's progression algorithm for LTL [BacchusKabanza96].
- Super-good algorithms for Pure-Past LTL [DeGiacomoFuggittiFavoritoRubin20],[BonassiDeGiacomoFuggittiGereviniScala22].
- State-of-the-art symbolic tablaux algorithms implemente in BLACK for Pure-Past LTL [GeattiGiganteMontanari21]

See Francesco Fuggitti's talk on Pure-Past LTL in planning!

Outline

Blurring of LTL$_f$ and LTL

Often, LTL$_f$ has been blurred with LTL, generating confusion in the AI and BPM literature of early 2000's.

From [Edelkamp2006]

"We can cast the Büchi automaton as an NFA, which accepts a word if it terminates in an accepting state."

From [GereviniEtAl2009]

"Since PDDL temporal constraints are normally evaluated over finite trajectories, the Büchi acceptance condition (an accepting state is visited infinitely often) reduces to the standard acceptance condition that the automaton is in an accepting state at the end of the trajectory."

From original DECLARE paper [VanDerAalstPesic06]:

We use the original (LTL) algorithm . . . , but . . . we specify that each execution of the model will eventually end.

- *We introduce an "invisible" activity end, which represents the ending activity in the model.*
- *We use this activity to specify that the service will end - the **termination** constraint. This constraint has the LTL formula $\Diamond end \land \Box(end \supset \bigcirc end)$."*
- *No other activity will hold when end holds*

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

These descriptions are misleading, and, if taken literally, wrong!

Blurring of LTL_f and LTL

Often, LTL_f has been blurred with LTL, generating confusion in the AI and BPM literature of early 2000's.

From [Edelkamp2006]

"We can cast the Büchi automaton as an NFA, which accepts a word if it terminates in an accepting state."

From [GereviniEtAl2009]

"Since PDDL temporal constraints are normally evaluated over finite trajectories, the Büchi acceptance condition (an accepting state is visited infinitely often) reduces to the standard acceptance condition that the automaton is in an accepting state at the end of the trajectory."

From original DECLARE paper [VanDerAalstPesic06]:

We use the original (LTL) algorithm . . . , but . . . we specify that each execution of the model will eventually end.

- *We introduce an "invisible" activity end, which represents the ending activity in the model.*
- *We use this activity to specify that the service will end - the* ***termination*** *constraint. This constraint has the LTL formula $\Diamond end \wedge \Box(end \supset \bigcirc end)$."*
- *No other activity will hold when end holds*

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

These descriptions are misleading, and, if taken literally, wrong!

Blurring of LTL_f and LTL

From [DeGiacomoDeMasellisMontali14]:

- Many LTL/LTL_f formulas (but not all) are insensitive to infiniteness: roughly speaking, even blurring the distinction between interpreting them on finite traces or on infinite traces, they maintain their meaning.
  - All DECLARE patterns, but one used rarely, are insensitive to infiniteness
  - Virtually all typical action domain specification in KR and Planning
  - All PDDL 3.0 trajectory constraints requiring at least one proposition to be true in propositional formulas, but (`always` ϕ)
- This, may help explaining why such wrong intuition has remained the basis for algorithms in systems for years.

Concerns about blurring infinite and finite traces was already rised by [BauerHaslum10], where correctness conditions are considered in extending finite traces by repeating at infinitum the propositional assignment in the last element of the finite trace.

Research rationale

While the blurring between infinite and finite traces has been of help as a jump start, AI and BPM are now sharpening focus on the finite trace assumption.

Blurring of LTL_f and LTL

From [DeGiacomoDeMasellisMontali14]:

- Many LTL/LTL_f formulas (but not all) are insensitive to infiniteness: roughly speaking, even blurring the distinction between interpreting them on finite traces or on infinite traces, they maintain their meaning.
  - All DECLARE patterns, but one used rarely, are insensitive to infiniteness
  - Virtually all typical action domain specification in KR and Planning
  - All PDDL 3.0 trajectory constraints requiring at least one proposition to be true in propositional formulas, but (`always` ϕ)
- This, may help explaining why such wrong intuition has remained the basis for algorithms in systems for years.

Concerns about blurring infinite and finite traces was already rised by [BauerHaslum10], where correctness conditions are considered in extending finite traces by repeating at infinitum the propositional assignment in the last element of the finite trace.

Research rationale

While the blurring between infinite and finite traces has been of help as a jump start, AI and BPM are now sharpening focus on the finite trace assumption.

Blurring of LTL_f and LTL

From [DeGiacomoDeMasellisMontali14]:

- Many LTL/LTL_f formulas (but not all) are insensitive to infiniteness: roughly speaking, even blurring the distinction between interpreting them on finite traces or on infinite traces, they maintain their meaning.
  - All DECLARE patterns, but one used rarely, are insensitive to infiniteness
  - Virtually all typical action domain specification in KR and Planning
  - All PDDL 3.0 trajectory constraints requiring at least one proposition to be true in propositional formulas, but (`always` ϕ)
- This, may help explaining why such wrong intuition has remained the basis for algorithms in systems for years.

Concerns about blurring infinite and finite traces was already rised by [BauerHaslum10], where correctness conditions are considered in extending finite traces by repeating at infinitum the propositional assignment in the last element of the finite trace.

Research rationale

While the blurring between infinite and finite traces has been of help as a jump start, AI and BPM are now sharpening focus on the finite trace assumption.

Outline

LTL_f and Automata

Key point

LTL_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 Finite-state Automata (AFA);
- 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!

LTL$_f$ and Automata

Key point

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

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

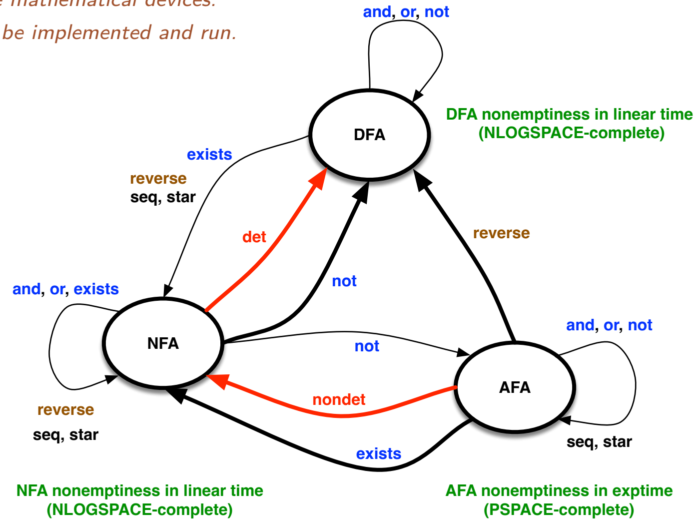
- in linear time if $\mathcal{A}_\varphi$ is an Alternating Finite-state Automata (AFA);
- in exponential time if $\mathcal{A}_\varphi$ is an Nondeterministic Finite-state Automaton (NFA);
- in double exponential time if $\mathcal{A}_\varphi$ is an Deterministic Finite-state Automaton (DFA).

We can compile reasoning into automata based procedures!

LTL$_f$ and Automata

Summary of automata theory on finite sequences:

- *NFA's and AFA's are mathematical devices.*
- *DFA's, instead, can be implemented and run.*



LTL_f and Automata

Key point

LTL_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 Finite-state Automata (AFA);
- 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!

Alternating Automata on Finite Words (AFA)

$\mathcal{A} = (2^{\mathcal{P}}, Q, q_0, \delta, F)$

- $2^{\mathcal{P}}$ alphabet
- Q is a finite nonempty set of states
- q_0 is the initial state
- F is a set of accepting states
- δ is a transition function $\delta : Q \times 2^{\mathcal{P}} \to B^+(Q)$, where $B^+(Q)$ is a set of positive boolean formulas whose atoms are states of Q.

AFA run

Given an input word $a_0, a_1, \ldots a_{n-1}$, an AFA run of an AFA is a tree (rather than a sequence) labelled by states of AFA such that

- root is labelled by q_0;
- if node x at level i is labelled by a state q and $\delta(q, a_i) = \Theta$, then either Θ is `true` or some $P \subseteq Q$ satisfies Θ and x has a child for each element in P.

A run is accepting if all leaves at depth n are labeled by states in F. Thus, a branch in an accepting run has to hit the `true` transition or hit an accepting state after reading all the input word $a_0, a_1, \ldots, a_{n-1}$.

(We adopt notation of "An Automata-Theoretic Approach to Linear Temporal Logic" by Moshe Vardi, 1996).

AFA $\mathcal{A}_\varphi$ associated with an LTL_f formula φ (in NNF)

$\mathcal{A}_\varphi = (2^\mathcal{P}, CL_\varphi, "\varphi", \delta, F)$ where

- $2^\mathcal{P}$ is the alphabet *($\mathcal{P}$ includes a special proposition $Last$ to denote the last element of the trace)*,
- CL_φ is the state set
- $"\varphi"$ is the initial state
- $F = \emptyset$ is the set of final states, which is empty
- δ is the transition function, defined as:

$$
\begin{array}{ll}
\delta("A", \Pi) & = \texttt{true} \text{ if } A \in \Pi \\
\delta("A", \Pi) & = \texttt{false} \text{ if } A \notin \Pi \\
\delta("\neg A", \Pi) & = \texttt{false} \text{ if } A \in \Pi \\
\delta("\neg A", \Pi) & = \texttt{true} \text{ if } A \notin \Pi \\
\delta("\varphi_1 \wedge \varphi_2", \Pi) & = \delta("\varphi_1", \Pi) \wedge \delta("\varphi_2", \Pi) \\
\delta("\varphi_1 \vee \varphi_2", \Pi) & = \delta("\varphi_1", \Pi) \vee \delta("\varphi_2", \Pi) \\
\delta("\bigcirc\varphi", \Pi) & = \begin{cases} "\varphi" & \text{if } Last \notin \Pi \\ \texttt{false} & \text{if } Last \in \Pi \end{cases} \\
\delta("\Diamond\varphi", \Pi) & = \delta("\varphi", \Pi) \ \vee \ \delta("\bigcirc\Diamond\varphi", \Pi) \\
\delta("\varphi_1 \,\mathcal{U}\, \varphi_2", \Pi) & = \delta("\varphi_2", \Pi) \ \vee \ (\delta("\varphi_1", \Pi) \wedge \delta("\bigcirc(\varphi_1 \,\mathcal{U}\, \varphi_2)", \Pi)) \\
\delta("\bullet\varphi", \Pi) & = \begin{cases} "\varphi" & \text{if } Last \notin \Pi \\ \texttt{true} & \text{if } Last \in \Pi \end{cases} \\
\delta("\Box\varphi", \Pi) & = \delta("\varphi", \Pi) \ \wedge \ \delta("\bullet\Box\varphi", \Pi) \\
\delta("\varphi_1 \,\mathcal{R}\, \varphi_2", \Pi) & = \delta("\varphi_2", \Pi) \ \wedge \ (\delta("\varphi_1", \Pi) \vee \delta("\bullet(\varphi_1 \,\mathcal{R}\, \varphi_2)", \Pi))
\end{array}
$$

Negation Normal Form for LTL_f

We put the LTL_f formula in NNF, because AFA's transitions return positive boolean combinations of states ($B^+(Q)$).

NNF

Negation Normal Form for LTL_f: for $a \in AP$

$$\varphi ::= \mathsf{true} \mid \mathsf{false} \mid A \mid \neg A \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \bigcirc\varphi \mid \bullet\varphi \mid \Diamond\varphi \mid \Box\varphi \mid \varphi\,\mathcal{U}\,\varphi \mid \varphi\mathcal{R}\varphi$$

Each LTL_f formula φ admits an equivalent in NNF denoted $nnf(\varphi)$, which is obtained in linear time in the size formula by pushing negation all the way, exploiting duals through the follow equivalence:

- $\neg\neg\varphi \equiv \varphi$
- $\neg(\varphi_1 \wedge \varphi_2) \equiv \neg\varphi_1 \vee \neg\varphi_2$
- $\neg(\varphi_1 \vee \varphi_2) \equiv \neg\varphi_1 \wedge \neg\varphi_2$
- $\neg\bigcirc\varphi \equiv \bullet\neg\varphi$
- $\neg\bullet\varphi \equiv \bigcirc\neg\varphi$
- $\neg\Diamond\varphi \equiv \Box\neg\varphi$
- $\neg\Box\varphi \equiv \Diamond\neg\varphi$
- $\neg(\varphi_1\,\mathcal{U}\,\varphi_1) \equiv \neg\varphi_1\mathcal{R}\neg\varphi_2$
- $\neg(\varphi_1\mathcal{R}\varphi_1) \equiv \neg\varphi_1\,\mathcal{U}\,\neg\varphi_2$

States of the AFA $\mathcal{A}_\varphi$

The states of $\mathcal{A}_\varphi$ are the subformulas of φ once expanded using the fixpoint equivalence.
This set of formulas is called the syntactic closure of φ.

Syntactic Closure of an LTL_f formula

The syntactic closure, aka "Fisher-Ladner closure", CL_φ of an LTL_f formula φ is a set of LTL_f formulas inductively defined as follows:

$$
\begin{array}{l}
\varphi \in CL_\varphi \\
\neg A \in CL_\varphi \text{ if } A \in CL_\varphi \\
A \in CL_\varphi \text{ if } \neg A \in CL_\varphi \\
\varphi_1 \wedge \varphi_2 \in CL_\varphi \text{ implies } \varphi_1, \varphi_2 \in CL_\varphi \\
\varphi_1 \vee \varphi_2 \in CL_\varphi \text{ implies } \varphi_1, \varphi_2 \in CL_\varphi \\
\bigcirc\varphi \in CL_\varphi \text{ implies } \varphi \in CL_\varphi \\
\Diamond\varphi \in CL_\varphi \text{ implies } \varphi, \bigcirc\Diamond\varphi \in CL_\varphi \\
\varphi_1 \,\mathcal{U}\, \varphi_2 \in CL_\varphi \text{ implies } \varphi_1, \varphi_2, \bigcirc(\varphi_1 \,\mathcal{U}\, \varphi_2) \in CL_\varphi \\
\bullet\varphi \in CL_\varphi \text{ implies } \varphi \in CL_\varphi \\
\Box\varphi \in CL_\varphi \text{ implies } \varphi, \bullet\Box\varphi \in CL_\varphi \\
\varphi_1 \,\mathcal{R}\, \varphi_2 \in CL_\varphi \text{ implies } \varphi_1, \varphi_2, \bullet(\varphi_1 \,\mathcal{R}\, \varphi_2) \in CL_\varphi
\end{array}
$$

Observe that the cardinality of CL_φ is linear in the size of φ.

LTL_f fixpoint equations

- $\Diamond\varphi \equiv \varphi \vee \bigcirc(\Diamond\varphi)$
- $\Box\varphi \equiv \varphi \wedge \bullet(\Box\varphi)$
- $\varphi_1 \,\mathcal{U}\, \varphi_2 \equiv \varphi_2 \vee (\varphi_1 \wedge \bigcirc(\varphi_1 \,\mathcal{U}\, \varphi_2))$
- $\varphi_1 \mathcal{R} \varphi_2 \equiv \varphi_2 \wedge (\varphi_1 \vee \bullet(\varphi_1 \mathcal{R} \varphi_2))$

LTL_f and Automata

AFA $\mathcal{A}_\varphi$ associated with an LTL_f formula φ (in NNF)

$\mathcal{A}_\varphi = (2^{\mathcal{P}}, CL_\varphi, "\varphi", \delta, F)$ where

- $2^{\mathcal{P}}$ is the alphabet,
- CL_φ is the state set,
- $"\varphi"$ is the initial state
- $F = \emptyset$ is the empty set of final states
- δ is the transition function

Transition function δ

$$
\begin{array}{ll}
\delta("A", \Pi) & = \texttt{true} \text{ if } A \in \Pi \\
\delta("A", \Pi) & = \texttt{false} \text{ if } A \notin \Pi \\
\delta("\neg A", \Pi) & = \texttt{false} \text{ if } A \in \Pi \\
\delta("\neg A", \Pi) & = \texttt{true} \text{ if } A \notin \Pi \\
\delta("\varphi_1 \wedge \varphi_2", \Pi) & = \delta("\varphi_1", \Pi) \wedge \delta("\varphi_2", \Pi) \\
\delta("\varphi_1 \vee \varphi_2", \Pi) & = \delta("\varphi_1", \Pi) \vee \delta("\varphi_2", \Pi) \\
\delta("\bigcirc\varphi", \Pi) & = \begin{cases} "\varphi" & \text{if } \mathit{Last} \notin \Pi \\ \texttt{false} & \text{if } \mathit{Last} \in \Pi \end{cases} \\
\delta("\Diamond\varphi", \Pi) & = \delta("\varphi", \Pi) \ \vee \ \delta("\bigcirc\Diamond\varphi", \Pi) \\
\delta("\varphi_1 \,\mathcal{U}\, \varphi_2", \Pi) & = \delta("\varphi_2", \Pi) \ \vee \ (\delta("\varphi_1", \Pi) \wedge \delta("\bigcirc(\varphi_1 \,\mathcal{U}\, \varphi_2)", \Pi)) \\
\delta("\bullet\varphi", \Pi) & = \begin{cases} "\varphi" & \text{if } \mathit{Last} \notin \Pi \\ \texttt{true} & \text{if } \mathit{Last} \in \Pi \end{cases} \\
\delta("\Box\varphi", \Pi) & = \delta("\varphi", \Pi) \ \wedge \ \delta("\bullet\Box\varphi", \Pi) \\
\delta("\varphi_1 \,\mathcal{R}\, \varphi_2", \Pi) & = \delta("\varphi_2", \Pi) \ \wedge \ (\delta("\varphi_1", \Pi) \vee \delta("\bullet(\varphi_1 \,\mathcal{R}\, \varphi_2)", \Pi))
\end{array}
$$

LTL_f fixpoint equations

- $\Diamond\varphi \equiv \varphi \vee \bigcirc(\Diamond\varphi)$
- $\Box\varphi \equiv \varphi \wedge \bullet(\Box\varphi)$
- $\varphi_1 \,\mathcal{U}\, \varphi_2 \equiv \varphi_2 \vee (\varphi_1 \wedge \bigcirc(\varphi_1 \,\mathcal{U}\, \varphi_2))$
- $\varphi_1 \mathcal{R} \varphi_2 \equiv \varphi_2 \wedge (\varphi_1 \vee \bullet(\varphi_1 \mathcal{R} \varphi_2))$

LTL$_f$ and Automata

AFAs can be transformed into NFA with standard algorithms in exponential time.

NFA $\mathcal{A}_\varphi$ associated with an LTL$_f$ formula φ (in NNF)

δ transition function

$$
\begin{aligned}
\delta("A", \Pi) &= \mathtt{true} \text{ if } A \in \Pi\\
\delta("A", \Pi) &= \mathtt{false} \text{ if } A \not\in \Pi\\
\delta("\neg A", \Pi) &= \mathtt{false} \text{ if } A \in \Pi\\
\delta("\neg A", \Pi) &= \mathtt{true} \text{ if } A \not\in \Pi\\
\delta("\varphi_1 \wedge \varphi_2", \Pi) &= \delta("\varphi_1", \Pi) \wedge \delta("\varphi_2", \Pi)\\
\delta("\varphi_1 \vee \varphi_2", \Pi) &= \delta("\varphi_1", \Pi) \vee \delta("\varphi_2", \Pi)\\
\delta("\bigcirc\varphi", \Pi) &= \begin{cases} "\varphi" & \text{if } \mathit{Last} \not\in \Pi\\ \mathtt{false} & \text{if } \mathit{Last} \in \Pi \end{cases}\\
\delta("\Diamond\varphi", \Pi) &= \delta("\varphi", \Pi) \;\vee\; \delta("\bigcirc\Diamond\varphi", \Pi)\\
\delta("\varphi_1 \,\mathcal{U}\, \varphi_2", \Pi) &= \delta("\varphi_2", \Pi) \;\vee\; (\delta("\varphi_1", \Pi) \wedge \delta("\bigcirc(\varphi_1 \,\mathcal{U}\, \varphi_2)", \Pi))\\
\delta("\bullet\varphi", \Pi) &= \begin{cases} "\varphi" & \text{if } \mathit{Last} \not\in \Pi\\ \mathtt{true} & \text{if } \mathit{Last} \in \Pi \end{cases}\\
\delta("\Box\varphi", \Pi) &= \delta("\varphi", \Pi) \;\wedge\; \delta("\bullet\Box\varphi", \Pi)\\
\delta("\varphi_1 \,\mathcal{R}\, \varphi_2", \Pi) &= \delta("\varphi_2", \Pi) \;\wedge\; (\delta("\varphi_1", \Pi) \vee \delta("\bullet(\varphi_1 \,\mathcal{R}\, \varphi_2)", \Pi))
\end{aligned}
$$

AFA2NFA transformation

algorithm LTL$_f$2NFA
input LTL$_f$ formula φ
output NFA $A_\varphi = (2^{\mathcal{P}}, \mathcal{S}, \{s_0\}, \varrho, \{s_f\})$
$s_0 \leftarrow \{"\varphi"\}$ ▷ single initial state
$s_f \leftarrow \emptyset$ ▷ single final state
$\mathcal{S} \leftarrow \{s_0, s_f\}$, $\varrho \leftarrow \emptyset$
while ($\mathcal{S}$ or ϱ change) **do**
  if($q \in \mathcal{S}$ and $q' \models \bigwedge_{("\psi" \in q)} \delta("\psi", \Pi)$)
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{q'\}$ ▷ update set of states
    $\varrho \leftarrow \varrho \cup \{(q, \Pi, q')\}$ ▷ update transition relation

Using function δ we can build the NFA A_φ of an LTL$_f$ formula φ in a forward fashion. States of A_φ are sets of atoms (recall that each atom is quoted φ subformulas) to be interpreted as a conjunction; the empty conjunction $\emptyset$ stands for $\mathtt{true}$.

LTL_f and automata

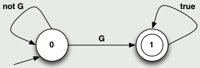
Key point

LTL_f formulas can be translated into a finite-state automaton on finite words $\mathcal{A}_\varphi$ such that:

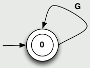
$$t \models \varphi \text{ iff } t \in \mathcal{L}(\mathcal{A}_\varphi)$$

- in linear time if $\mathcal{A}_\varphi$ is an Alternating Finite-state Automata (AFA);
- 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).

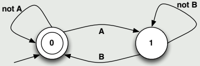
Example (Automata for some LTL_f formulas)



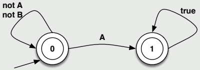
$\Diamond G$



$\Box G$



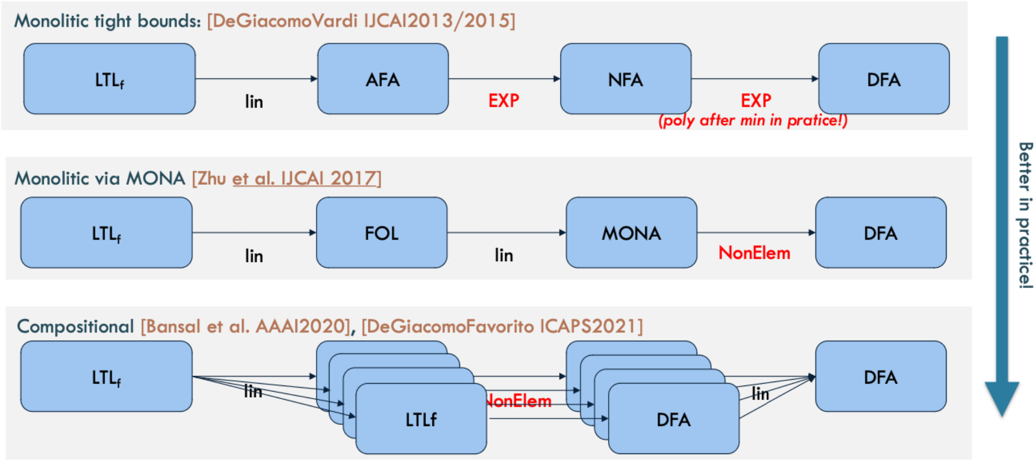
$\Box(A \supset \bigcirc\Diamond B)$



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

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

LTL_f to Automata Techniques



LTL_f to Automata Techniques

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

$$\text{LTL}_f \xrightarrow{\text{lin}} \text{AFA} \xrightarrow{\text{EXP}} \text{NFA} \xrightarrow[\text{EXP}]{\text{PDDL/on the fly}} \text{DFA}$$

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

$$\text{LTL}_f \xrightarrow[\text{exploit formula semantics to obtain DFA on the fly in forward fashion}]{} \text{DFA}$$

Based on "next normal form", i.e., fixpoint equations, as in [BacchusKabanzaAAAI1996]:

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)

Outline

LTL$_f$ Reasoning

LTL$_f$ Satisfiability (φ SAT)

1: Given LTL$_f$ formula φ
2: Compute AFA for φ *(linear)*
3: Compute corresponding NFA *(exponential)*
4: Check NFA for nonemptiness *(NLOGSPACE)*
5: Return result of check

LTL$_f$ Validity (φ VAL)

1: Given LTL$_f$ formula φ
2: Compute AFA for $\neg\varphi$ *(linear)*
3: Compute corresponding NFA *(exponential)*
4: Check NFA for nonemptiness *(NLOGSPACE)*
5: Return complemented result of check

LTL$_f$ Logical Implication ($\Gamma \models \varphi$)

1: Given LTL$_f$ formulas Γ and φ
2: Compute AFA for $\Gamma \wedge \neg\varphi$ *(linear)*
3: Compute corresponding NFA *(exponential)*
4: Check NFA for nonemptiness *(NLOGSPACE)*
5: Return complemented result of check

Thm: All the above reasoning tasks are PSPACE-complete. (On-the-fly construction of NFA while checking nonemptiness.)

As for the infinite traces.

LTL_f Model Checking

Given a transition system $\mathcal{T}$, check that **all finite executions allowed by $\mathcal{T}$** satisfy an LTL_f specification φ.

LTL_f model checking algorithm

1: Given Transition System $\mathcal{T}$ and LTL_f formula φ
2: Compute the NFA $A_\mathcal{T}$ of $\mathcal{T}$ *(linear in $\mathcal{T}$, in fact constant!)*
3: Compute AFA for $\neg\varphi$ *(linear in φ)*
4: Compute corresponding NFA $\mathcal{A}_{\neg\varphi}$ *(exponential in φ)*
5: Compute NFA $A_\mathcal{T} \times \mathcal{A}_{\neg\varphi}$ for $(\mathcal{A}_\mathcal{T} \wedge \mathcal{A}_{\neg\varphi})$ *(polynomial)*
6: Check resulting NFA $A_\mathcal{T} \times \mathcal{A}_{\neg\varphi}$ for nonemptiness *(NLOGSPACE)*
7: Return complemented result of check

Thm: Verification is PSPACE-complete, and most importantly polynomial in the transition system.

The same results holds for LTL on infinite traces.

Model Checking

Question: What kind of model checking properties can be expressed in LTL_f?

Answer: All LTL safety properties! (And nothing else.)

Maybe LTL_f *is not that interesting for model checking ... but it is super-interesting for synthesis!*

(Remember we are interested in AI agents!!!)

Model Checking

Question: What kind of model checking properties can be expressed in LTL_f?

Answer: All LTL safety properties! (And nothing else.)

Maybe LTL_f is not that interesting for model checking ... but it is super-interesting for synthesis!

(Remember we are interested in AI agents!!!)

Model Checking

Question: What kind of model checking properties can be expressed in LTL$_f$?

Answer: All LTL safety properties! (And nothing else.)

Maybe LTL$_f$ is not that interesting for model checking ... but it is super-interesting for synthesis!

(Remember we are interested in AI agents!!!)

Model Checking

Question: What kind of model checking properties can be expressed in LTL_f?

Answer: All LTL safety properties! (And nothing else.)

Maybe LTL_f is not that interesting for model checking ... but it is super-interesting for synthesis!

(Remember we are interested in AI agents!!!)

Model Checking

Question: What kind of model checking properties can be expressed in LTL_f?

Answer: All LTL safety properties! (And nothing else.)

Maybe LTL_f *is not that interesting for model checking ... but it is super-interesting for synthesis!*

(Remember we are interested in AI agents!!!)

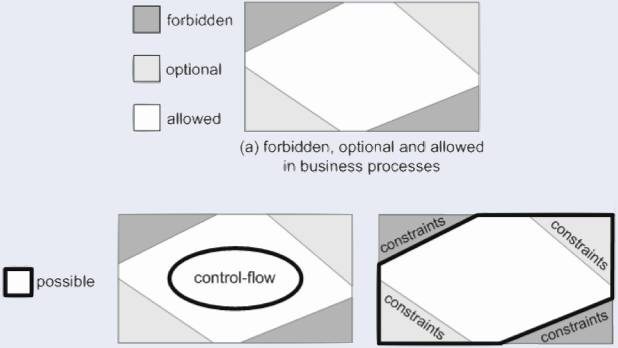
Outline

LTL$_f$ Synthesis Under Full Controllability (BPM)

This is a first, very simple, form of program synthesis!

Synthesis under full controllability

Given declarative specification in terms of LTL$_f$ constraints, extract process/program/domain description/transition system that captures exactly specification.



(a) forbidden, optional and allowed in business processes

(b) procedural workflow

(c) declarative workflow

(From DECLARE [PesicBovsnavkiDraganVanDerAalst10])

LTL_f Synthesis Under Full Controllability (BPM)

Process corresponding to LTL_f specification always exists for finite traces!

Any LTL_f specification correspond to exactly one process: *the* corresponding minimal DFA!

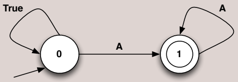
```
Given LTLf formula φ
    Compute AFA for φ (linear in φ)
    Compute corresponding NFA (exponential in φ)
    Compute corresponding DFA (exponential in NFA)
    Trim DFA to avoid dead ends (polynomial)
    Optional: Minimize DFA (polynomial)
Return resulting DFA
```

IMPORTANT

- This is a BEAUTIFUL RESULT: We go from purely declarative to fully procedural!
- It relies on the possibility of obtaining a deterministic automaton, a DFA, which is a machine, and hence a process. [AbadiLamportWolper89]
- Does NOT hold in the infinite trace settings!

Example (Over infinite traces the following LTL formulas do not correspond to any process)

Consider the LTL formula $\Diamond\Box A$ and its Büchi Automaton:



Outline

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, controllers, robots, etc.)
    [Church1963], [AbadiLamportWolper1989], [PnueliRosner1989]

Reactive Synthesis

- Reactive synthesis is equipped with a elegant and comprehensive theory
  [Finkbeiner2018],[EhlersLafortuneTripakisVardi2017]
- Reactive synthesis is conceptually related to planning in nondeterministic domains
  [DeGiacomoVardi2015], [DeGiacomoRubin2018], [CamachoMuiseBaierMcIlraith2018]

Program Synthesis

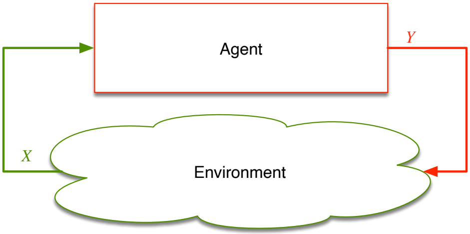
Program Synthesis

- **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, controllers, robots, etc.)
    [Church1963], [AbadiLamportWolper1989], [PnueliRosner1989]

Reactive Synthesis

- Reactive synthesis is equipped with a elegant and comprehensive theory
  [Finkbeiner2018],[EhlersLafortuneTripakisVardi2017]
- Reactive synthesis is conceptually related to planning in nondeterministic domains
  [DeGiacomoVardi2015], [DeGiacomoRubin2018], [CamachoMuiseBaierMcIlraith2018]

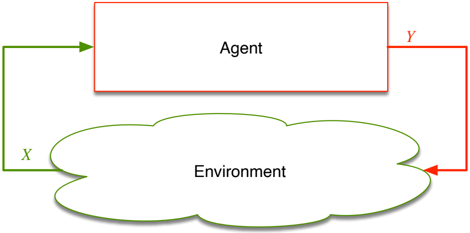
Agent in Environment



Inputs and outputs

- The agent receives input $\mathcal{X}$ from the environment.
- The agent sends output $\mathcal{Y}$ to the environment.
- Input $\mathcal{X}$ can be fluents, features, program input, etc.
- Output $\mathcal{Y}$ can be actions, control instructions, program outputs, etc.
- Input is uncontrollable by the agent (it is under the control of the environment).
- Output is controllable by the agent.

Synthesis as a Game



Game View

Agent is playing a game with environment, with the LTL_f/LTL specifications being the winning condition.

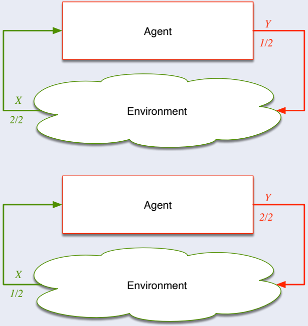
- Agent chooses controllable output $Y \in 2^{\mathcal{Y}}$
- Environment chooses uncontrollable input $X \in 2^{\mathcal{X}}$
- Round: agent and environment set their values
- Play: finite trace τ over $(\mathcal{X} \cup \mathcal{Y})$
- Agent decides when to stop
- Specification: LTL_f formula φ
- Agent wins $\tau \models \varphi$

Synthesis as a Game

Game Rounds

Agent is playing a game with environment, with the LTL_f/LTL specifications being the winning condition.

- Agent chooses controllable output $Y \in 2^{\mathcal{Y}}$
- Environment chooses uncontrollable input $X \in 2^{\mathcal{X}}$
- Round: agent and environment set their values
  - Pair output and resulting input (agent action and environment reaction)
  - Pair input and next output (environment state and next agent action)
- Play: finite trace τ over $(\mathcal{X} \cup \mathcal{Y})$
- Agent decides when to stop
- Specification: LTL_f formula φ
- Agent wins $\tau \models \varphi$



Pair actions and states in a time instant (reminder)

Decide how we need to pair actions and states in a time instant

- Pair the agent action and the resulting state, (in fact labeling of the state) of the environment

  The propositional representation a for an action a will stand for "action a just executed".

- Pair current environment (labeling of the) state and the next action instructed by the agent

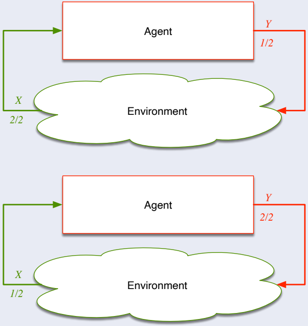
  The propositional representation a for an action a will stand for "action a just instructed to be executed next".

Synthesis as a Game

Game Rounds

Agent is playing a game with environment, with the LTL_f/LTL specifications being the winning condition.

- Agent chooses controllable output $Y \in 2^{\mathcal{Y}}$
- Environment chooses uncontrollable input $X \in 2^{\mathcal{X}}$
- Round: agent and environment set their values
  - Pair output and resulting input $\Longleftarrow$
    (agent action and environment reaction)
  - Pair input and next output
    (environment state and next agent action)
- Play: finite trace τ over $(\mathcal{X} \cup \mathcal{Y})$
- Agent decides when to stop
- Specification: LTL_f formula φ
- Agent wins $\tau \models \varphi$



Pair actions and states in a time instant (reminder)

Decide how we need to pair actions and states in a time instant

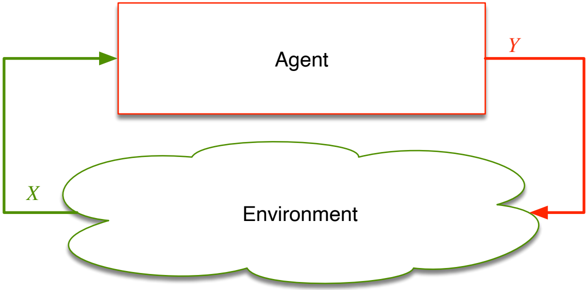
- Pair the agent action and the resulting state, (in fact labeling of the state) of the environment $\Longleftarrow$

  The propositional representation a for an action a will stand for "action a just executed".

- Pair current environment (labeling of the) state and the next action instructed by the agent

  The propositional representation a for an action a will stand for "action a just instructed to be executed next".

Agent strategie



Agent strategies

Agent strategy *(also called, "plan", "policy", "protocol", "process", "program", "behavior")*:

$$\sigma_a : (2^{\mathcal{X}})^* \to 2^{\mathcal{Y}}$$

where

- $(2^{\mathcal{X}})^*$ denotes the history of inputs observed so far by the agent

  (a finite sequence of fluents configurations)

- $2^{\mathcal{Y}}$ denotes the next output of the agent

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

Synthesis from LTL$_f$ Specifications

Given a LTL$_f$ formula φ over a set $\mathcal{P}$ of propositions partitioned into two disjoint sets:

- $\mathcal{X}$ controlled by environment
- $\mathcal{Y}$ controlled by agent

Find an agent strategy σ_a to set the values of $\mathcal{Y}$ in such a way that for all possible values of $\mathcal{X}$, controlled by the environment, the LTL$_f$ formula φ can be made true.

Algorithm for LTL$_f$ synthesis

1: Given LTL$_f$ formula φ
2: Compute AFA 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: LTL$_f$ synthesis is 2-EXPTIME-complete.

Same as for infinite traces

DFA Games

DFA games

A DFA game $\mathcal{G} = (2^{\mathcal{X} \cup \mathcal{Y}}, S, s_0, \delta, F)$, is such that:

- $\mathcal{X}$ controlled by environment; $\mathcal{Y}$ controlled by agent;
- $2^{\mathcal{X} \cup \mathcal{Y}}$, alphabet of game;
- S, states of game;
- s_0, initial state of game;
- $\delta : S \times 2^{\mathcal{X} \cup \mathcal{Y}} \to S$, transition function of the game: given current state s and a choice of propositions X and Y the resulting state of the game is $\delta(s, (X, Y)) = s'$;
- F, final states of game, where game can be considered terminated.

Winning condition for DFA games

Let

$$PreAdv(\mathcal{E}) = \{s \in S \mid \exists Y \in 2^{\mathcal{Y}}.\forall X \in 2^{\mathcal{X}}.\delta(s, (X, Y)) \in \mathcal{E}\}$$

Compute the set $Win(\mathcal{G})$ of winning states of a DFA game $\mathcal{G}$, i.e., states from which the agent can win the DFA game $\mathcal{G}$, by least-fixpoint:

- $Win_0(\mathcal{G}) = F$ (the final states of $\mathcal{G}$)
- $Win_{i+1}(\mathcal{G}) = Win_i(\mathcal{G}) \cup PreAdv(Win_i(\mathcal{G}))$
- $Win(\mathcal{G}) = \bigcup_i Win_i(\mathcal{G})$

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

DFA Games

DFA games

A DFA game $\mathcal{G} = (2^{\mathcal{X} \cup \mathcal{Y}}, S, s_0, \delta, F)$, is such that:

- $\mathcal{X}$ controlled by environment; $\mathcal{Y}$ controlled by agent;
- $2^{\mathcal{X} \cup \mathcal{Y}}$, alphabet of game;
- S, states of game;
- s_0, initial state of game;
- $\delta : S \times 2^{\mathcal{X} \cup \mathcal{Y}} \to S$, transition function of the game: given current state s and a choice of propositions X and Y the resulting state of the game is $\delta(s, (X, Y)) = s'$;
- F, final states of game, where game can be considered terminated.

Winning condition for DFA games

Let

$$PreAdv(\mathcal{E}) = \{s \in S \mid \exists Y \in 2^{\mathcal{Y}}.\forall X \in 2^{\mathcal{X}}.\delta(s, (X, Y)) \in \mathcal{E}\}$$

Compute the set $Win(\mathcal{G})$ of winning states of a DFA game $\mathcal{G}$, i.e., states from which the agent can win the DFA game $\mathcal{G}$, by least-fixpoint:

- $Win_0(\mathcal{G}) = F$ (the final states of $\mathcal{G}$)
- $Win_{i+1}(\mathcal{G}) = Win_i(\mathcal{G}) \cup PreAdv(Win_i(\mathcal{G}))$
- $Win(\mathcal{G}) = \bigcup_i Win_i(\mathcal{G})$

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

Computing Strategies

To actually compute a strategy, we need to

- Apply any choice function to get only one value $choice(\omega(s))$ (any choice would be good) among those in $\omega(s)$, where

$$\omega(s) = \{Y \mid \text{ if } s \in Win_{i+1}(\mathcal{G}) - Win_i(\mathcal{G}) \text{ then } \forall X.\delta(s,(X,Y)) \in Win_i(\mathcal{G})\}$$

- Compute the corresponding strategy $\sigma_a : (2^{\mathcal{X}})^* \to 2^{\mathcal{Y}}$ via a transducer $\mathcal{T}_G$ obtained form the game $\mathcal{G}$ and the function $choice(\omega(s))$.

The obtained σ_a is memory-full, but has only finite number of states.

Strategy as a transducer

The strategy returned is a transducer $\mathcal{T}_{\mathcal{G}} = (2^{\mathcal{X}}, S, s_0, \varrho, \omega_{choice})$ where:

- $2^{\mathcal{X}}$ is the alphabet of the trasducer;
- S are the states of the trasducer;
- s_0 is the initial state;
- $\varrho : S \times 2^{\mathcal{X}} \to S$ is the transition function (partial) such that

$$\varrho(s,X) = \delta(s,(X, choice(\omega(s))))$$

- $\omega_{choice} : S \to 2^{\mathcal{Y}}$ is the output function such that

$$\omega_{choice} = choice(\omega(s))$$

Synthesis in LTL

Synthesis for general LTL specifications does not scale *(yet)*.

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)

Outline

Planning and Synthesis

Planning in Nondeterministic Domain

- Fluents $\mathcal{F}$ (propositions) – controlled by the environment
- Actions $\mathcal{A}$ (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 as game between two players

- Arena: the domain
- Players: agent and environment
- Game: agent tries to force eventually reaching G no matter how other environment reacts
- Problem: find agent-strategy $\sigma_a : (2^{\mathcal{F}})^* \to \mathcal{A}$ to win the game

Complexity

EXPTIME-complete in size of domain specified in PDDL.

See Sheila McIlraith's and Alberto Camacho's talks!

Synthesis

- Inputs $\mathcal{X}$ (propositions) – controlled by the environment
- Outputs $\mathcal{Y}$ (propositions) – controlled by the agent
- Domain – not considered
- Goal φ – arbitrary LTL_f (or other temporal logic specification) on both $\mathcal{X}$ and $\mathcal{Y}$

Synthesis as game between two players

- Arena: unconstraint! clique among all possible assignments for $\mathcal{X}$ and $\mathcal{Y}$
- Players: agent and environment
- Game: agent tries to force a play that satisfies φ no matter how other environment reacts.
- Problem: find agent-strategy $\sigma_a (2^{\mathcal{X}})^* \to 2^{\mathcal{Y}}$ to win the game.

Complexity

2EXPTIME-complete in size of φ.

Planning and Synthesis

Planning in Nondeterministic Domain

- Fluents $\mathcal{F}$ (propositions) – controlled by the environment
- Actions $\mathcal{A}$ (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 as game between two players

Arena: the domain

- Players: agent and environment
- Game: agent tries to force eventually reaching G no matter how other environment reacts
- Problem: find agent-strategy $\sigma_a : (2^{\mathcal{F}})^* \to \mathcal{A}$ to win the game

Complexity

EXPTIME-complete in size of domain specified in PDDL.

Synthesis

- Inputs $\mathcal{X}$ (propositions) – controlled by the environment
- Outputs $\mathcal{Y}$ (propositions) – controlled by the agent
- Domain – not considered
- Goal φ – arbitrary LTL_f (or other temporal logic specification) on both $\mathcal{X}$ and $\mathcal{Y}$

Synthesis as game between two players

Arena: unconstraint! clique among all possible assignments for $\mathcal{X}$ and $\mathcal{Y}$

- Players: agent and environment
- Game: agent tries to force a play that satisfies φ no matter how other environment reacts.
- Problem: find agent-strategy $\sigma_a(2^{\mathcal{X}})^* \to 2^{\mathcal{Y}}$ to win the game.

Complexity

2EXPTIME-complete in size of φ.

Planning and Synthesis

Planning in Nondeterministic Domain

- Fluents $\mathcal{F}$ (propositions) – controlled by the environment
- Actions $\mathcal{A}$ (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 as game between two players

Arena: the domain

- Players: agent and environment
- Game: agent tries to force eventually reaching G no matter how other environment reacts
- Problem: find agent-strategy $\sigma_a : (2^{\mathcal{F}})^* \to \mathcal{A}$ to win the game

Complexity

EXPTIME-complete in size of domain specified in PDDL.

Synthesis

- Inputs $\mathcal{X}$ (propositions) – controlled by the environment
- Outputs $\mathcal{Y}$ (propositions) – controlled by the agent
- Domain – not considered
- Goal φ – arbitrary LTL_f (or other temporal logic specification) on both $\mathcal{X}$ and $\mathcal{Y}$

Synthesis as game between two players

Arena: unconstraint! clique among all possible assignments for $\mathcal{X}$ and $\mathcal{Y}$

- Players: agent and environment
- Game: agent tries to force a play that satisfies φ no matter how other environment reacts.
- Problem: find agent-strategy $\sigma_a(2^{\mathcal{X}})^* \to 2^{\mathcal{Y}}$ to win the game.

Complexity

2EXPTIME-complete in size of φ.

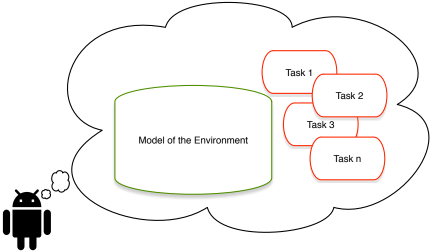
We want to revisit the assumption that the environment is unconstrained!

Outline

Planning in Nondeterministic Domains (FOND_{sp})

Planning in nondeterministic domains

- Environment Model (DOM)
  - Environment model is called "domain"
  - Specs of environment's behaviors of the world in response to agent's action
  - Domain expressed as with specific formalisms
    - PDDL
  - DOM is, or better generates, a non-deterministic transition system, i.e., a **game arena for two players Agent and Env**!
- Agent Task (GOAL)
  - Agent task is called "goal"
  - Specs of task to achieve
  - GOAL expressed as reaching a state of the domain with desired properties
- Find agent plan/program/strategy/policy that fulfills GOAL in DOM



Find plan that fulfills the desired task
in spite of how the environment responds,
i.e., wins the GOAL in nondeterministic DOM

Planning in Nondeterministic Domains

Who controls what?

Fluents controlled by **environment**

Actions controlled by **agent**

Observe: $\delta(s, a, s')$

Game arena induced by a nondeterministic domain

If we consider this information on the control, then T_D is in fact a **game arena**: $T_D = (2^{\mathcal{F}}, \mathcal{A}, s_0, \alpha, \delta)$ where:

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

Hence to execute a transition in state s

- The agent needs to choose the action a
- The environment need to choose the resulting state s'.

Planning in Nondeterministic Domain

Planning

Given a nondeterministic domain D and a goal G in propositional logic:

- Find agent executable strategy (or plan) σ_a such that for every environment strategies σ_e that are compliant with D, we have that the $play(\sigma_a, \sigma_e) = s_0, a_1, s_1, \ldots, s_{n-1}, a_n, s_n$ is such that $s_n \models G$.

In other words find a strategy (or plan) σ_a that reaches a state where G holds no matter what the environment does.

The strategy σ_a, if it exists is called **winning strategy**

LTL_f Goals

(1) In order express arbitrary goals in LTL_f –or LTL–, we need to:

Represent actions as propositions

Decide how we represent actions as propositions of LTL_f formulas.

- Use one proposition a for each action a. Then:
  - We need to add the requirement that at most one action proposition is true in each instant $\Box(a \supset \bigwedge_{b \in A \wedge b \neq a} \neg b)$.
- Use a binary (logarithmic) encoding of action each a. Then:
  - Each action a is represented as a boolean formula a over the propositions for the binary encoding;
  - Some binary encoding will correspond to non-existing actions, if the number of actions is not a power of 2. In this case we need to specify what these spurious action do in the transition system, e.g., $nope$, or we need to forbid them.

For now, we will adopt the first way of representing actions, but later when we study symbolic technique we will also use the latter.

(cf. LTL_f Model Checking)

LTL_f Goals

(2) In order express arbitrary goals in LTL_f –or LTL–, we also need to:

Pair actions and states in a time instant

Decide how we need to pair actions and states in a time instant

- Pair the agent action and the resulting state, (in fact labeling of the state) of the environment

  The propositional representation a for an action a will stand for "action a just executed".

- Pair current environment (labeling of the) state and the next action instructed by the agent

  The propositional representation a for an action a will stand for "action a just instructed to be executed next".

Both ways of pairing actions and states are fine. But choosing one or the other is essential, because it changes how we specify properties in LTL_f.

(cf. LTL_f Model Checking)

LTL$_f$ Goals

LTL$_f$-traces

A T_D trace $e_0, a_1, e_1, \cdots, a_n, e_n$ induces a corresponding LTL$_f$-trace:

- If we pair action and the resulting state: $(dummy, e_0), (a_1, e_1), \cdots, (a_n, e_n)$, where $dummy$ is a dummy starting action.
- If we pair state and the next action: $(e_0, a_1), (e_1, a_2) \cdots, (e_{n-1} a_n), (e_n, dummy)$, where $dummy$ is a dummy ending action.

Example

The way we pair actions and states changes how we specify properties in LTL$_f$:
Suppose we want to say:

every time that ϕ_1 is true in the current state if we do action a we get ϕ_2 in the next state".

- If we pair action and the resulting state, we write: $\square(\phi_1 \supset \bullet(a \supset \phi_2))$
- If we pair state and the next action, we write: $\square((\phi_1 \wedge a) \supset \bullet\phi_2)$

*In this course we pair **action and the resulting state** to have traces that represents cleanly histories (things already happened).*

LTL_f Goals

LTL_f-traces

A T_D trace $e_0, a_1, e_1, \cdots, a_n, e_n$ induces a corresponding LTL_f-trace:

- If we pair action and the resulting state: $(dummy, e_0), (a_1, e_1), \cdots, (a_n, e_n)$, where $dummy$ is a dummy starting action.
- If we pair state and the next action: $(e_0, a_1), (e_1, a_2) \cdots, (e_{n-1} a_n), (e_n, dummy)$, where $dummy$ is a dummy ending action.

Example

The way we pair actions and states changes how we specify properties in LTL_f:
Suppose we want to say:

every time that ϕ_1 is true in the current state if we do action a we get ϕ_2 in the next state".

- If we pair action and the resulting state, we write: $\Box(\phi_1 \supset \bullet(a \supset \phi_2))$
- If we pair state and the next action, we write: $\Box((\phi_1 \land a) \supset \bullet\phi_2)$

In this course we pair action and the resulting state to have traces that represents cleanly histories (things already happened).

LTL_f Goals

LTL_f-traces

A T_D trace $e_0, a_1, e_1, \cdots, a_n, e_n$ induces a corresponding LTL_f-trace:

- If we pair action and the resulting state: $(dummy, e_0), (a_1, e_1), \cdots, (a_n, e_n)$, where $dummy$ is a dummy starting action.
- If we pair state and the next action: $(e_0, a_1), (e_1, a_2) \cdots, (e_{n-1} a_n), (e_n, dummy)$, where $dummy$ is a dummy ending action.

Example

The way we pair actions and states changes how we specify properties in LTL_f:
Suppose we want to say:

every time that ϕ_1 is true in the current state if we do action a we get ϕ_2 in the next state".

- If we pair action and the resulting state, we write: $\Box(\phi_1 \supset \bullet(a \supset \phi_2))$
- If we pair state and the next action, we write: $\Box((\phi_1 \wedge a) \supset \bullet\phi_2)$

In this course we pair action and the resulting state to have traces that represents cleanly histories (things already happened).

Nondeterministic Domains as Automata

With these decisions taken we can transform the nondeterministic domain $T_D = (2^{\mathcal{F}}, \mathcal{A}, s_0, \alpha, \delta)$ into an automaton recognizing all its traces.

Automaton A_D for D is a DFA!!!

$A_D = (2^{\mathcal{F} \cup \mathcal{A}}, Q, q_{init}, \rho, F)$ where:

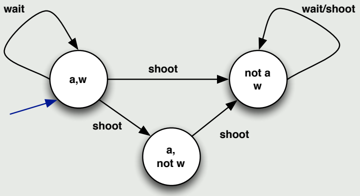
- $2^{\mathcal{F} \cup \mathcal{A}}$ alphabet (actions $\mathcal{A}$ include dummy $start$ action)
- $Q = 2^{\mathcal{F}} \cup \{q_{init}\}$ set of states
- q_{init} dummy initial state
- $F = 2^{\mathcal{F}}$ (all states of the domain are final)
- $\rho(s, [a, s']) = s'$ **with** $a \in \alpha(s)$**,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

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

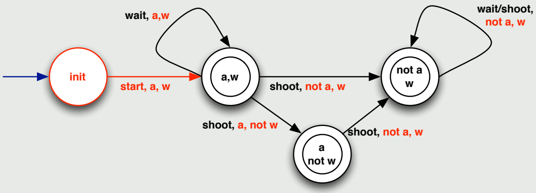
Nondeterministic Domains as Automata

Example (Simplified Yale shooting domain variant)

- Domain $\mathcal{T}_D$:



- DFA A_D:



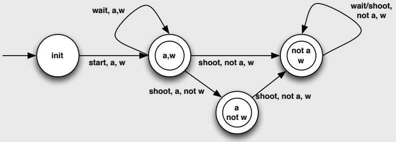
Nondeterministic Domains as Automata

Planning in nondeterministic domains

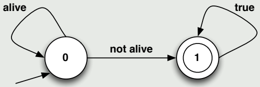
- Set the **arena** formed by all traces that satisfy both the DFA A_D for D and the DFA for $\Diamond G$ where G is the goal.
- Compute a **winning strategy**. *(EXPTIME-complete in D, constant in G)*

Example (Simplified Yale shooting domain)

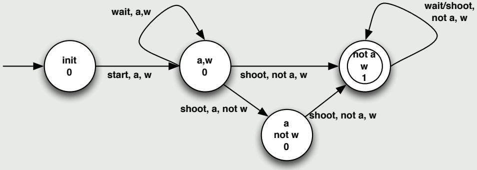
A_D



$A_{\Diamond\neg a}$



$A_D \cap A_{\Diamond\neg a}$:



strategy

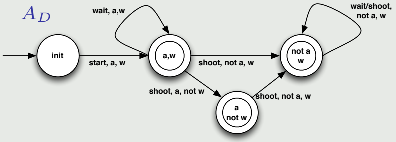
$init, 0$	$\rightarrow$	$start$
$a, w, 0$	$\rightarrow$	$shoot$
$a, \neg w, 0$	$\rightarrow$	$shoot$
$\neg a, w, 1$	$\rightarrow$	win!

FOND_{sp} for LTL_f Goals

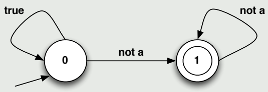
Example (Simplified Yale shooting domain)

Consider the goal $\Diamond\Box\neg a$.

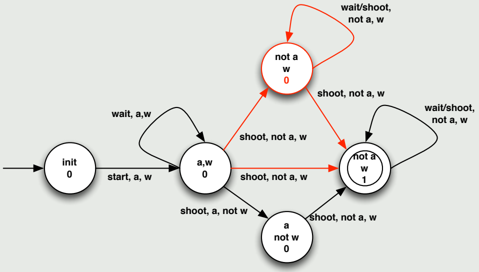
A_D



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



$A_D \cap A_{\Diamond\Box\neg a}$:



Can we use directly NFA's?

No, because of a basic mismatch

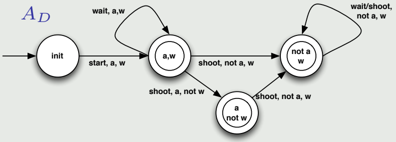
- NFA have perfect **foresight**, or **clairvoyance** *(angelic nondeterminism)*
- Strategies must be runnable: **depend only on past**, not future *(devilish nondeterminism)*

FOND_{sp} for LTL_f Goals

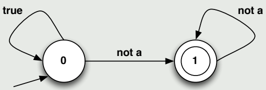
Example (Simplified Yale shooting domain)

Consider the goal $\Diamond\Box\neg a$.

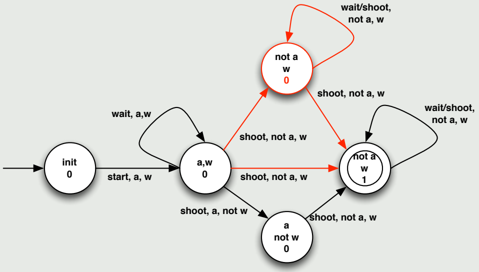
A_D



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



$A_D \cap A_{\Diamond\Box\neg a}$:



Can we use directly NFA's?

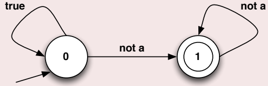
No, because of a basic mismatch

- NFA have perfect **foresight**, or **clairvoyance** *(angelic nondeterminism)*
- Strategies must be runnable: **depend only on past**, not future *(devilish nondeterminism)*

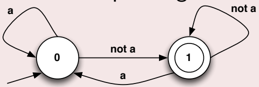
FOND$_{sp}$ for LTL$_f$ Goals

We need first to determinize the NFA for LTL$_f$ formula

NFA for $\Diamond\Box\neg a$



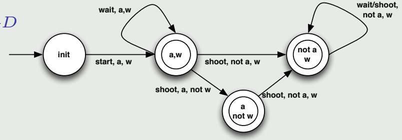
corresponding DFA



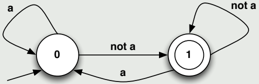
(DFA can be exponential in NFA in general)

Example (Simplified Yale shooting domain)

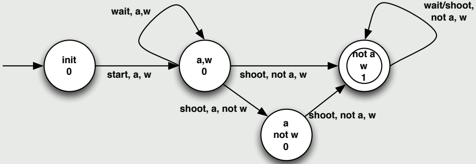
A_D



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



$A_D \cap A_{\Diamond\Box\neg a}$:



strategy

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

FOND_{sp} for LTL_f Goals

FOND_{sp} for LTL_f goals

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

1: Given LTL_f domain D and goal φ
2: Compute NFA for φ (exponential)
3: Determinize NFA to DFA (exponential)
4: Compute intersection with DFA of D (polynomial)
5: Synthesize winning strategy for DFA game (linear)
6: Return strategy

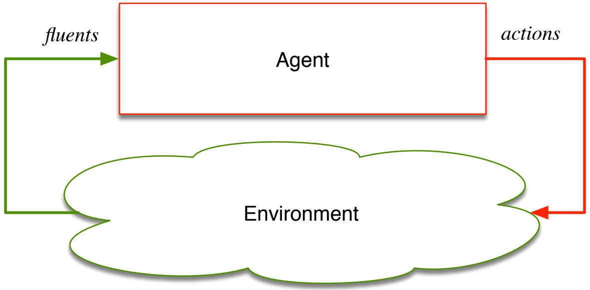
Theorem

Planning in nondetermnistic domains for LTL_f goals is:

- *EXPTIME-complete in the domain (compactly represented using of fluents – polynomial in number of states);*
- *2-EXPTIME-complete in the goal.*

Outline

Planning revisited: Synthesis with a model of the environment



Task

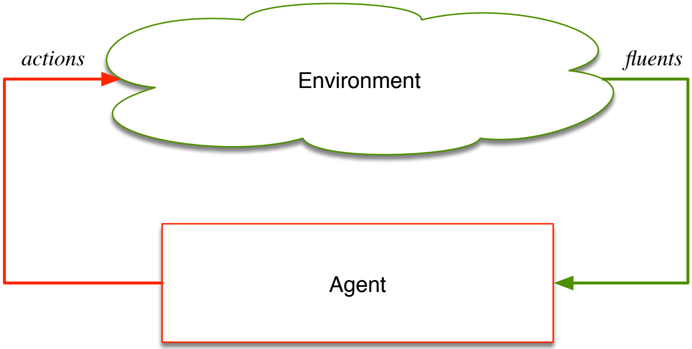
Task for the agent, also called goal

- It is a specification of the good traces (i.e., plays generated by agent and env that are considered good.)
- Typically of a simpler form wrt synthesis, i.e.,:

$$\textit{eventually}\ GoodStateOfAffairs \ \wedge\ \textit{always}\,\neg PreViolated$$

- But goal can also be temporally extended, i.e., arbitrary formulas expressed in LTL_f

Planning revisited: Synthesis with a model of the environment



Domain

- Planning consider 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, that is:

A specification of how the environment responds to agent actions.

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

Planning in nondeterministic domains

Transition system induced by a nondeterministic domain

A nondeterministic domain $D = (\mathcal{F}, \mathcal{A}, I)$ induces a transition system $T_D = (2^{\mathcal{F}}, \mathcal{A}, s_0, \alpha, \delta)$ where:

- $\mathcal{F}$ is the set of fluents (atomic propositions)
- $\mathcal{A}$ is the set of actions (atomic symbols)
- $2^{\mathcal{F}}$ is the set of states
- s_0 is the initial state (initial assignment to fluents)
- $\alpha(s) \subseteq \mathcal{A}$ represents action preconditions

  Note: fulfilling action precondition is a responsibility of the agent!
- $\delta(s, a, s')$ with $a \in \alpha(s)$ represents nondeterministic action effects (including frame).

  Note: fulfilling action effects + frame is a responsibility of the environment!

Remove action preconditions form the domain

In the following we assume action have no preconditions in the domains.

- When preconditions are not satisfied the environment remains in the same state. *–requires conditional effects*
- In traces/plays, "agent can select an action only if its satisfies its precondition" can be expressed in the (LTL_f) goal.
- Indeed: if $Pre(a) = \varphi_a$ then it suffices to require in the goal $\Box((\bigcirc a) \supset \varphi_a$, i.e.:

  "Always, if the action a has been just executed next, then its precondition φ_a is now satisfied."

In this way, the domain becomes a specification of the environment.

Planning in nondeterministic domains

Transition system induced by a nondeterministic domain

A nondeterministic domain $D = (\mathcal{F}, \mathcal{A}, I)$ induces a transition system $T_D = (2^{\mathcal{F}}, \mathcal{A}, s_0, \alpha, \delta)$ where:

- $\mathcal{F}$ is the set of fluents (atomic propositions)
- $\mathcal{A}$ is the set of actions (atomic symbols)
- $2^{\mathcal{F}}$ is the set of states
- s_0 is the initial state (initial assignment to fluents)
- $\alpha(s) \subseteq \mathcal{A}$ represents action preconditions

  Note: fulfilling action precondition is a responsibility of the agent!
- $\delta(s, a, s')$ with $a \in \alpha(s)$ represents nondeterministic action effects (including frame).

  Note: fulfilling action effects + frame is a responsibility of the environment!

Remove action preconditions form the domain

In the following we assume action have no preconditions in the domains.

- When preconditions are not satisfied the environment remains in the same state. *–requires conditional effects*
- In traces/plays, "agent can select an action only if its satisfies its precondition" can be expressed in the (LTL_f) goal.
- Indeed: if $Pre(a) = \varphi_a$ then it suffices to require in the goal $\Box((\bigcirc a) \supset \varphi_a$, i.e.:

  "Always, if the action a has been just executed next, then its precondition φ_a is now satisfied."

In this way, the domain becomes a specification of the environment.

Planning revisited: Synthesis with a model of the environment

Domain as a specification of the environment

$$[\![D]\!] = \{\sigma_e \mid \sigma_e \text{ complies with domain } D\}$$

where, an environment strategy σ_e complies with domain D, if for every trace $a_0, s_0, a_1, s_1, \cdots, a_n, s_n$ of T_D (a_0 is the dummy $start$ action) we have that $\sigma_e(a_0, , \ldots, a_{n+1}) = s'_{n+1}$ is such that $\delta(s_n, a_{n+1}, s_{n+1})$.

Planning in nondeterministic domains

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

- Realizability:

$$\text{check if } \exists \sigma_a . \forall \sigma_e \in [\![D]\!] . trace(\sigma_a, \sigma_e) \models Goal$$

- Synthesis:

$$\text{find } \sigma_a \text{ such that } \forall \sigma_e \in [\![D]\!] . trace(\sigma_a, \sigma_e) \models Goal$$

Synthesis + environment model = planning

We can transfer the idea of working with a model of the world to synthesis

Synthesis for a task in an environment

Given a task $Task$ for the agent, and a specification Env of the possible environment strategies:

- Realizability:

$$\text{check if } \exists \sigma_a . \forall \sigma_e \in [\![Env]\!] . trace(\sigma_a, \sigma_e) \models Task$$

- Synthesis:

$$\text{find } \sigma_a \text{ such that } \forall \sigma_e \in [\![Env]\!] . trace(\sigma_a, \sigma_e) \models Task$$

Specifying possible environment behaviors in LTL

- Can we use LTL/LTL$_f$ to specify an environment, i.e., the possible environment strategies?
- Yes, 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.

Consistent environment specifications

Is any LTL/LTLf formula a valid environment specification? No, Env needs to be "consistent"!:

$$[\![Env]\!] \neq \emptyset \qquad \text{i.e. } \exists \sigma_e.\forall \sigma_a.trace(\sigma_a, \sigma_e) \models Env$$

For example "eventually agent does action dec"

$$\textit{eventually } dec$$

is not a valid specification of the environment, since the agent might decide not to do dec.

Specifying possible environment behaviors in LTL

- Can we use LTL/LTL$_f$ to specify an environment, i.e., the possible environment strategies?
- Yes, 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.

Consistent environment specifications

Is any LTL/LTLf formula a valid environment specification? No, Env needs to be "consistent"!:

$$[\![Env]\!] \neq \emptyset \qquad \text{i.e. } \exists \sigma_e.\forall \sigma_a.trace(\sigma_a, \sigma_e) \models Env$$

For example "eventually agent does action dec"

$$\mathit{eventually}\ dec$$

is not a valid specification of the environment, since the agent might decide not to do dec.

Specifying possible environment behaviors in LTL

- Can we use LTL/LTL$_f$ to specify an environment, i.e., the possible environment strategies?
- Yes, 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.

Consistent environment specifications

Is any LTL/LTLf formula a valid environment specification? No, Env needs to be "consistent"!:

$$[\![Env]\!] \neq \emptyset \qquad \text{i.e. } \exists \sigma_e . \forall \sigma_a . trace(\sigma_a, \sigma_e) \models Env$$

For example "eventually agent does action dec"

eventually dec

is not a valid specification of the environment, since the agent might decide not to do dec.

Specifying possible environment behaviors in LTL

- Can we use LTL/LTL$_f$ to specify an environment, i.e., the possible environment strategies?
- Yes, 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.

Consistent environment specifications

Is any LTL/LTLf formula a valid environment specification? No, Env needs to be "consistent"!:

$$[\![Env]\!] \neq \emptyset \qquad \text{i.e. } \exists \sigma_e.\forall \sigma_a.trace(\sigma_a, \sigma_e) \models Env$$

For example "eventually agent does action dec"

eventually dec

is not a valid specification of the environment, since the agent might decide not to do dec.

Check consistency of environment specifications

Check consistency of environment specifications

First we need to check **consistency** of the environment specification Env, i.e., our model of the world ($[\![Env]\!] \neq \emptyset$). We can do so my checking **unrealizability** of agent task Env (we exploit determinacy of resulting games):

$$\exists \sigma_a . \forall \sigma_e . trace(\sigma_a, \sigma_e) \models \neg Env$$

Theorem

Checking consistency of LTL/LTL$_f$ *environment specification is 2EXPTIME-complete.*

Note: for LTL *we could solve realizability for the environment directly.*
Instead for LTL$_f$ *no, because the agent, and not the environment, has the choice of stopping the trace*
– though we could solve a safety game instead of a reachability one (see later).

How to solve synthesis with world models

Solve synthesis

Theorem ([AminofDeGiacomoMuranoRubinICAPS2019])

Let $Task$ *be a agent task and* Env *be a consistent* $\mathrm{LTL}_f/\mathrm{LTL}$ *environment specification. Then*

- *There is* ***agent strategy realizing*** $Task$ *in* Env *iff there is an* ***agent strategy realizing*** $Env \supset Task$

$$\exists\sigma_a.\forall\sigma_e \in [\![Env]\!].trace(\sigma_a,\sigma_e) \models Task \ \textit{ iff } \ \exists\sigma_a.\forall\sigma_e.trace(\sigma_a,\sigma_e) \models Env \supset Task$$

- *Moreover,* ***every agent strategy realizing*** $Env \supset Task$ *is a* ***agent strategy realizing*** $Task$ *in* Env

$$\textit{for all } \sigma_a \textit{ we have: } \forall\sigma_e.trace(\sigma_a,\sigma_e) \models Env \supset Task \ \textit{ implies } \ \forall\sigma_e \in [\![Env]\!].trace(\sigma_a,\sigma_e) \models Task$$

but not viceversa!

Hence, to find **agent strategy realizing** $Task$ under the environment specification Env, we can use standard $\mathrm{LTL}/\mathrm{LTL}_f$ synthesis for

$$Env \supset Task$$

Theorem

Solving $\mathrm{LTL}/\mathrm{LTL}_f$ *synthesis under environment specification is 2EXPTIME-complete.*

Environment specification in $\mathrm{LTL}/\mathrm{LTL}_f$

- FOND planning for LTLf tasks
  - Strong: these are simple Markovian Safety properties [DeGiacomoRubinIJCAI2018]
  - Stochastic fairness: as FOND strong cyclic planning, but on an arena that is obtained from domain D and Task [DeGiacomoRubinIJCAI2018], [Aminof et al. ICAPS 2020]

- Env: Safe, coSafe, GR(1), Live
  - Env = Safe: Safe implies Task iff not Safe or Task. But not Safe is LTLf so this is LTLf synthesis
  - Env = Simple Fairness and Stability: Use task to generate arena, then play for single nested fixpoint [Zhu et al. AAAI2020]
  - Env = Safe & coSafe: reduction to deterministic Buchi automata [Camachio et al 2018], use Safe, coSafe and Task to generate arena, then play for single nested fixpoint [De Giacomo et al. KR2020]
  - Env = Safe & GR(1): reduction to GR(1), use Task and Safe to generate arena, then play GR(1) game (double nested fixpoint) [De Giacomo et al. IJCAI2021]
  - Env = Live & Safe: reduction to Live implies LTLf, solvable by LTL synthesis, needed for (hopefully small) Live [De Giacomo et al. KR 2020]

- Env = Live & Safe + agent MUST stop!
  - Agent stops env irrelevant: drop Live, and solve Safe implies Task (LTLf synthesis) [De Giacomo et al KR2021]
  - When agent stops env can continue to evolve: the agent cannot act anymore, though some AgtSafe must be maintained! Find by model checking "agent safe states" where AgtStafe can be maintained without doing anything, then solve Safe implies Task& "at agent safe states" (LTLf synthesis) [De Giacomo et al KR2021]

Outline

Conclusion

- We have looked at impact of expressing temporal properties/constraints/goals on traces that are finite as typical in AI Planning and BPM modeling.
- By the way, this assumption has been considered a sort of accident in much of the AI and BPM literatures, and standard temporal logics (on infinite traces) have been hacked to fit this assumption, with some success, but later clean solutions have been devised.
- We have focussed on LTL_f on finite traces, which has the expressive power of FOL, but we can go to full MSO (monadic second-order logic over finite traces) at no cost, e.g., by LDL_f which is a nice combination of LTL_f and RE, and behaves computational as LTL_f.
- We can also look at Pure Past LTL, which has exactly the same expressive power of LTL_f, but whose DFA's are only 1EXPTIME (due to a property of reverse languages).
- There are elegant and effective techniques for reasoning, verification and execially synthesis in this setting – Logics-Automata-Games in this case it's not "just theory".