

Master in Artificial Intelligence and Robotics (AIRO)

Electives in AI Reasoning Agents

Fabio Patrizi

Sapienza University of Rome, Italy
patrizi@diag.uniroma1.it

A.Y. 2021-2022

Linear Temporal Logics on Finite Traces: LTL_f and LDL_f *

* slides based on G. De Giacomo's (www.diag.uniroma1.it/degiacomo)

- 1 Motivation
- 2 LTL_f : LTL on Finite Traces
- 3 LTL_f : Expressive Power
- 4 LDL_f : Linear Dynamic Logic on Finite Traces
- 5 LTL_f/ LDL_f Reasoning and Verification
- 6 LTL_f/ LDL_f Program Synthesis
- 7 Conclusion

- 1 Motivation
- 2 LTL_f : LTL on Finite Traces
- 3 LTL_f : Expressive Power
- 4 LDL_f : Linear Dynamic Logic on Finite Traces
- 5 LTL_f/LDL_f Reasoning and Verification
- 6 LTL_f/LDL_f Program Synthesis
- 7 Conclusion

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

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

Temporal extended goals and constraints in AI

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

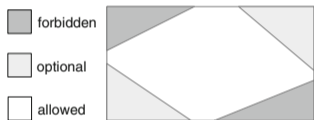
However:

- Often, LTL is interpreted on **finite** trajectories/traces.
- Often, distinction between interpreting LTL on **infinite** or on **finite** traces is **blurred**.

- 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]

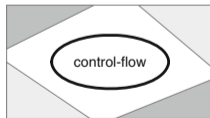
Business Process Management community has proposed a declarative approach to business process modeling based on LTL on finite traces: DECLARE

Basic idea: Drop explicit representation of processes, and LTL formulas specify the allowed **finite traces**.
[VanDerAalstPestic06] [PesticBovsnavkiDraganVanDerAalst10].

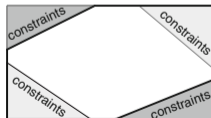


(a) forbidden, optional and allowed in business processes

 possible



(b) procedural workflow













(c) declarative workflow

Motivation: BPM – DECLARE patterns

DECLARE promotes the use of a controlled set of notable LTL formulas on (finite traces) for process specification. [VanDerAalstPesic06]

Example (Main DECLARE Patterns)

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

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

- 1 Motivation
- 2 **LTL_f: LTL on Finite Traces**
- 3 LTL_f: Expressive Power
- 4 LDL_f: Linear Dynamic Logic on Finite Traces
- 5 LTL_f/LDL_f Reasoning and Verification
- 6 LTL_f/LDL_f Program Synthesis
- 7 Conclusion

LTL_f: the language

$$\varphi ::= A \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \bigcirc\varphi \mid \varphi_1 \mathcal{U} \varphi_2$$

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

Main formal properties:

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

Assuming finite or infinite traces has **big impact**.

Example

Consider the following formula:

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

- On **infinite** traces:



- On **finite** traces:



Interpreting LTL on infinite or finite traces has **big impact**.

Example

Consider the following formula:

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

- On **infinite** traces:



- On **finite** traces:

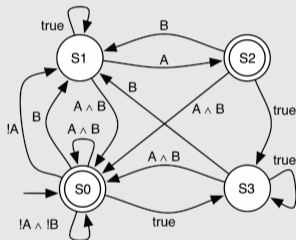


Interpreting LTL on infinite or finite traces has **big impact**.

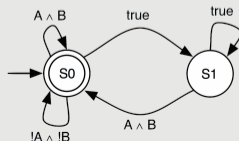
Example

Consider again the formula: $\Box(A \supset \Diamond B) \wedge \Box(B \supset \Diamond A)$

- Buchi automaton accepting its **infinite** traces:



- NFA accepting its **finite** traces:



Example (Unintuitive LTL_f formulas - "Response")

$$\square \diamond A$$

for any point in the trace there is a point later where A holds ("Response").

- On **infinite** traces:



- On **finite** traces becomes equivalent to **last point in the trace satisfies A** , i.e. $\diamond(\text{Last} \wedge A)$



Example (Unintuitive LTL_f formulas - "Persistence")

$$\diamond \square \varphi$$

there exists a point in the trace such that from then on φ holds ("Persistence").

- On **infinite** traces:



- On **finite** traces becomes equivalent to **last point in the trace satisfies** φ , i.e. $\diamond (Last \wedge \varphi)$



*In other words, no direct nesting of **eventually** and **always** connectives is meaningful in LTL_f , this contrast what happens in LTL of infinite traces.*

Example (Unintuitive LTL_f formulas)

- $\square\lozenge\varphi$: for any point in the trace there is a point later where φ holds (“Response”).
But this is equivalent to say that the **last point in the trace satisfies** φ , i.e.:

$$\lozenge(\text{Last} \wedge \varphi).$$

Notice that this meaning is completely different from the meaning on infinite traces and cannot be considered a “fairness” property as “response” is in the infinite case.

- $\lozenge\square\varphi$: there exists a point in the trace such that from then on φ holds (“Persistence”).
But again this is equivalent to say that the **last point in the trace satisfies** φ , i.e.:

$$\lozenge(\text{Last} \wedge \varphi).$$

*In other words, no direct nesting of **eventually** and **always** connectives is meaningful in LTL_f , this contrast what happens in LTL of infinite traces.*

Example (Capturing STRIPS Planning as LTL_f SAT)

- For each operator/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 LTL_f formula is SAT.

Example (Propositional SitCalc Basic Action Theories in LTL_f)

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

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

- **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**”:

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

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.

- 1 Motivation
- 2 LTL_f : LTL on Finite Traces
- 3 LTL_f : Expressive Power
- 4 LDL_f : Linear Dynamic Logic on Finite Traces
- 5 LTL_f/ LDL_f Reasoning and Verification
- 6 LTL_f/ LDL_f Program Synthesis
- 7 Conclusion

LTL_f can express any First-Order formula FOL over finite sequences.

x

FOL over finite sequences (aka $FO[<]$)

- First-order language formed by:
 - One binary predicate $<$: denoting total ordering between points in sequence;
 - Unary predicate symbols A : denoting points in sequence where a certain property A holds.
- Notice that with $<$ one can define:
 - $succ(x, y) \doteq (x < y) \wedge \neg \exists z. x < z < y$: denote the **successor** relation;
 - $x = y \doteq \forall z. x < z \equiv y < z$: denotes **equality** between points
 - 0 , the **initial** point, can be defined as that x such that $\neg \exists y. succ(y, x)$;
 - $last$, the **last** point, can be defined as that x such that $\neg \exists y. succ(x, y)$.

LTL_f to FOL

We can translate any LTL_f formula to FOL

- $fol(A, x) = A(x)$
- $fol(\neg\varphi, x) = \neg fol(\varphi, x)$
- $fol(\varphi \wedge \varphi', x) = fol(\varphi, x) \wedge fol(\varphi', x)$
- $fol(\bigcirc\varphi, x) = \exists y. succ(x, y) \wedge fol(\varphi, y)$
- $fol(\varphi \mathcal{U} \varphi', x) = \exists y. x \leq y \leq last \wedge fol(\varphi', y) \wedge \forall z. x \leq z < y \rightarrow fol(\varphi, z)$

Example

$\Box(LowPwr \supset \Diamond Recharged)$ is translated to
 $\forall x. LowPwr(x) \supset \exists y. x \leq y \leq last \wedge Recharged(y)$ [Kamp68]

And viceversa!

Theorem ([GabbayPnueliShelahStavi80] – see also [Kamp68])

LTL_f has the same expressive power of FOL.

LTL_f can express any FOL formula over finite sequences (and viceversa).

Can we do better?

Monadic Second-Order Logic (MSO) over finite sequences

MSO is a **strict extension** of the FOL language introduced above, where

- we add the possibility of writing formulas of the form

- $\forall X.\varphi$
- $\exists X.\varphi$

where X is a monadic (i.e., unary) **predicate variable** and φ may include atoms whose predicate is such variable.

- Binary predicates and constants remain exactly those introduced for FOL.

Reasoning in MSO over finite sequences is decidable, though nonelementary.

Theorem (Büchi-Elgot-Trakhtenbrot [Buchi60, Elgot61, Trakh62])

MSO on finite sequences has exactly the expressive power of Regular Expressions.

RE: Regular Expressions as temporal logic on finite traces

RE expressions are defined as follows:

$$\varrho ::= \phi \mid \varrho_1 + \varrho_2 \mid \varrho_1; \varrho_2 \mid \varrho^*$$

where ϕ is a propositional formula.

Reasoning in RE

- A trace t **satisfies** a RE expression ϱ iff $t \in \mathcal{L}(\varrho)$.
- A RE expression ϱ is **satisfiable** iff $\mathcal{L}(\varrho) \neq \emptyset$
- A RE expression ϱ is **valid** iff $\mathcal{L}(\varrho) = \Sigma^*$.

$\mathcal{L}(\varrho)$ is the language associated to ϱ .

Example

- “Safety” ($\Box\varphi$):

$$\varphi^*$$

that means that always, until the end of the trace, φ holds.

- “Liveness” ($\Diamond\varphi$):

$$\text{true}^*; \varphi; \text{true}^*$$

that means that eventually before the end of the trace φ holds.

- “Conditional response” ($\Box(\psi \supset \Diamond\varphi)$):

$$\overline{\overline{(\text{true}^*; \psi \wedge \neg\varphi; \overline{(\text{true}^*; \varphi; \text{true}^*)})}}$$

that means whenever ψ holds then later φ holds.

- “Ordered occurrence”:

$$\text{true}^*; \varphi_1; \text{true}^*; \varphi_2; \text{true}^*$$

that says φ_1 and φ_2 will both happen in order.

Example (Interesting RE expressions in BPM [DiCiccioMecella12])

Constraint	Regular expression	Example
Existence constraints		
$Existence(n, a)$	$[\bar{a}]^*(a[\bar{a}]^*)^{\{n, \infty\}}[\bar{a}]^*$	
$Participation(a) \equiv Existence(1, a)$	$[\bar{a}]^*(a[\bar{a}]^*)+[\bar{a}]^*$	<u>bc</u> aac
$Absence(m + 1, a)$	$[\bar{a}]^*(a[\bar{a}]^*)^{\{0, m\}}[\bar{a}]^*$	
$Uniqueness(a) \equiv Absence(2, a)$	$[\bar{a}]^*(a)?[\bar{a}]^*$	bcac
$Init(a)$	$a.*$	accbbababab
$End(a)$	$.*a$	bcaccbbabab
Relation constraints		
$RespondedExistence(a, b)$	$[\bar{a}]^*((a.*b) (b.*a))*[\bar{a}]^*$	<u>bc</u> aaccbbabab
$Response(a, b)$	$[\bar{a}]^*(a.*b)*[\bar{a}]^*$	<u>bc</u> aaccbbab
$AlternateResponse(a, b)$	$[\bar{a}]^*(a[\bar{a}]^*b)*[\bar{a}]^*$	bcaccbbab
$ChainResponse(a, b)$	$[\bar{a}]^*(ab[\bar{a}^b])^*[\bar{a}]^*$	bcabbbab
$Precedence(a, b)$	$[\bar{b}]^*(a.*b)*[\bar{b}]^*$	caaccbbabab
$AlternatePrecedence(a, b)$	$[\bar{b}]^*(a[\bar{b}]^*b)*[\bar{b}]^*$	caaccbaba
$ChainPrecedence(a, b)$	$[\bar{b}]^*(ab[\bar{a}^b])^*[\bar{b}]^*$	<u>cab</u> aba
$CoExistence(a, b)$	$[\bar{a}^b]^*((a.*b) (b.*a))*[\bar{a}^b]^*$	<u>bc</u> aaccbbabab
$Succession(a, b)$	$[\bar{a}^b]^*(a.*b)*[\bar{a}^b]^*$	caaccbbab
$AlternateSuccession(a, b)$	$[\bar{a}^b]^*(a[\bar{a}^b]^*b)*[\bar{a}^b]^*$	caccbab
$ChainSuccession(a, b)$	$[\bar{a}^b]^*(ab[\bar{a}^b])^*[\bar{a}^b]^*$	<u>cab</u> ab
Negative relation constraints		
$NotChainSuccession(a, b)$	$[\bar{a}]^*(a[\bar{a}^b] [\bar{a}]^*)^*([\bar{a}]^* a)$	<u>bc</u> aaccbbabab
$NotSuccession(a, b)$	$[\bar{a}]^*(a[\bar{b}]^*)^*[\bar{a}^b]^*$	bc <u>ac</u> ca
$NotCoExistence(a, b)$	$[\bar{a}^b]^*((a[\bar{b}]^*) (b[\bar{a}]^*))?$	ca <u>ac</u> ca

(Mostly RE translation of DECLARE LTL_f patterns)

Theorem ([McNaughtonPapert1971])

FOL on finite sequences has the same expressive power as *star-free* RE.

Star-free regular expressions

$$\varrho ::= \phi \mid \varrho_1 + \varrho_2 \mid \varrho_1; \varrho_2 \mid \bar{\varrho}$$

where $\bar{\varrho}$ stands for the complement of ϱ , i.e., $\mathcal{L}(\bar{\varrho}) = (2^{\mathcal{P}})^* / \mathcal{L}(\varrho)$.

Star-free regular expressions are strictly less expressive than RE since they do not allow for unrestrictedly expressing properties involving the Kleene star $$, which appears implicitly only to generate the universal language used in complementation.*

Example (Some interesting star-free regular expressions)

- $(2^{\mathcal{P}})^* = \text{true}^*$ is in fact star-free, as it can be expressed as $\overline{\text{false}}$
- $\text{true}^*; \phi; \text{true}^*$ is star-free, as true^* is star-free.
- ϕ^* for a propositional ϕ is also star-free, as it is equivalent to $\overline{\text{true}^*; \neg\phi; \text{true}^*}$.

Corollary

LTL_f has exactly the same expressive power as star-free RE.

Example (LTL_f constructs as star-free regular expressions)

- $\diamond\phi$ can be expressed as $\text{true}^*; \phi; \text{true}^*$
- $\square\phi$ can be expressed as $\overline{(\text{true}^*; \neg\phi; \text{true}^*)}$
- $\phi_1 \mathcal{U} \phi_2$ can be expressed as $\overline{(\text{true}^*; \neg\phi_1; \text{true}^*)}; \phi_2; \text{true}^*$

LTL_f is less expressive than RE!

Rationale

- LTL_f has the same expressive power as FOL, which is that of star-free RE.
- RE have the same expressive power as MSO, which is strictly higher than LTL_f .

Should we use RE instead of LTL_f ?

- RE as expressive as MSO – *good*
- But RE is not closed under negation and conjunction (they require to deeply transform the expression!) – *bad*
- Moreover, negation requires an exponential blow up! – *bad*

NO

Any better logic?

Is there a logic with the expressive power of MSO and RE, but which is as intuitive as LTL_f , possibly maintaining the same computational characteristics?

YES

- 1 Motivation
- 2 LTL_f : LTL on Finite Traces
- 3 LTL_f : Expressive Power
- 4 LDL_f : Linear Dynamic Logic on Finite Traces**
- 5 LTL_f/ LDL_f Reasoning and Verification
- 6 LTL_f/ LDL_f Program Synthesis
- 7 Conclusion

LDL_f: Linear Dynamic Logic on Finite Traces

- Directly inspired by the syntax of PDL [FisherLadner79], which is possibly the most well-known (propositional) logic of programs in CS.

(But now interpreted over finite traces.)

- Enhances LTL_f by including regular expressions in the temporal formulas.
In the infinite trace setting, such enhancement strongly advocated by industrial model checking [ForSpec02,PSL06].

LDL_f [DeGiacomoVardi13]

Syntax:

$$\varphi ::= \text{tt} \mid \text{ff} \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \langle \rho \rangle \varphi \mid [\rho] \varphi \quad \rho ::= \phi \mid \varphi? \mid \rho_1 + \rho_2 \mid \rho_1; \rho_2 \mid \rho^*$$

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

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

LTL_f can be translated into LDL_f in linear time

- $f(A) = \langle A \rangle \text{tt}$
- $f(\neg\varphi) = \neg f(\varphi)$
- $f(\varphi_1 \wedge \varphi_2) = f(\varphi_1) \wedge f(\varphi_2)$
- $f(\bigcirc\varphi) = \langle \text{true} \rangle f(\varphi)$
- $f(\varphi_1 \mathcal{U} \varphi_2) = \langle (f(\varphi_1)?; \text{true})^* \rangle f(\varphi_2)$

RE can be translated into LDL_f in linear time

$$g(\varrho) = \langle \varrho \rangle \text{end}$$

where *end* stands for “the traces ends”, i.e., $\langle \text{true} \rangle \text{ff}$.

Also, LDL_f can itself be translated into RE, though in exponential time.

Theorem ([DeGiacomoVardi13])

LDL_f has the same expressive power as RE and MSO on finite traces.

Example (AI procedural constraints – GOLOG)

Formalisms like GOLOG [Reiter01] can be used for expressing “procedural” temporal constraints/goals in AI [BaierFritzMcIlraith07]

GOLOG – propositional/finite domain variant

$$\delta ::= A \mid \varphi? \mid \delta_1 + \delta_2 \mid \delta_1; \delta_2 \mid \delta^* \mid \pi x. \delta(x) \mid \text{if } \phi \text{ then } \delta_1 \text{ else } \delta_2 \mid \text{while } \phi \delta$$

- $\pi x. \delta(x)$ stands for $\sum_{o \in Obj} \delta(o)$
- **if** ϕ **then** δ_1 **else** δ_2 stands for $(\phi?; \delta_1) + (\neg\phi?; \delta_2)$
- **while** ϕ **do** δ stands for $(\phi?; \delta)^*; \neg\phi?$

- $\langle \delta \rangle \phi$ in LDL_f captures the following SitCalc formula:

$$\exists s'. Do(\delta, s, s') \wedge s \leq s' \leq last \wedge \phi(s').$$

- $[\delta] \phi$ in LDL_f captures the following SitCalc formula:

$$\forall s'. Do(\delta, s, s') \wedge s \leq s' \leq last \supset \phi(s').$$

($\phi(s)$ “uniform” in s .)

Example

- “All coffee requests from person p will eventually be served”:

$$\Box(\text{request}_p \supset \Diamond \text{coffee}_p) \quad [\text{true}^*](\text{request}_p \supset \langle \text{true}^* \rangle \text{coffee}_p)$$

- “Every time the robot opens door d it closes it immediately after”:

$$\Box(\text{openDoor}_d \supset \bigcirc \text{closeDoor}_d) \quad [\text{true}^*](\langle \text{openDoor}_d \rangle \text{closeDoor}_d)$$

- “Before entering restricted area a the robot must have permission for a ”:

$$\neg \text{inArea}_a \cup \text{getPerm}_a \vee \Box \neg \text{inArea}_a \quad \langle (\neg \text{inArea}_a)^* \rangle \text{getPerm}_a \vee [\text{true}^*] \neg \text{inArea}_a$$

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

$$\langle (\neg \text{inArea}_a^*; \text{getPerm}_a; \neg \text{inArea}_a^*; \text{inArea}_a; \text{inArea}_a^*)^*; \neg \text{inArea}_a^* \rangle \text{end}$$

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

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

- 1 Motivation
- 2 LTL_f : LTL on Finite Traces
- 3 LTL_f : Expressive Power
- 4 LDL_f : Linear Dynamic Logic on Finite Traces
- 5 LTL_f/ LDL_f Reasoning and Verification**
- 6 LTL_f/ LDL_f Program Synthesis
- 7 Conclusion

Key point

Both LTL_f (and LDL_f) formulas can be translated in **linear time** to Alternating Automata on Finite Words (AFW).

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

where \mathcal{A}_φ is the AFW φ is translated into.

We can compile reasoning into automata based procedures!

Alternating Automata on Finite Words (AFW)

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

- $2^{\mathcal{P}}$ 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 $\delta : Q \times 2^{\mathcal{P}} \rightarrow B^+(Q)$, where $B^+(Q)$ is a set of positive boolean formulas whose atoms are states of Q .

AFW run

Given an input word a_0, a_1, \dots, a_{n-1} , an AFW run of an AFW is a tree (rather than a sequence) labelled by states of AFW 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 .

(We adopt notation of [Vardi96].)

To define the AFW \mathcal{A}_φ associated with an LT_L_f formula φ (in NNF), we need first to introduce its syntactic closure.

Syntactic Closure of an LT_L_f formula

The syntactic closure, also called Fisher-Ladner closure, CL_φ of an LT_L_f formula φ is a set of LD_L_f formulas inductively defined as follows:

- $\varphi \in CL_\varphi$
- $\neg A \in CL_\varphi$ if $A \in CL_\varphi$
- $A \in CL_\varphi$ if $\neg A \in CL_\varphi$
- $\varphi_1 \wedge \varphi_2 \in CL_\varphi$ implies $\varphi_1, \varphi_2 \in CL_\varphi$
- $\varphi_1 \vee \varphi_2 \in CL_\varphi$ implies $\varphi_1, \varphi_2 \in CL_\varphi$
- $\bigcirc \varphi \in CL_\varphi$ implies $\varphi \in CL_\varphi$
- $\diamond \varphi \in CL_\varphi$ implies $\varphi, \bigcirc \diamond \varphi \in CL_\varphi$
- $\varphi_1 \mathcal{U} \varphi_2 \in CL_\varphi$ implies $\varphi_1, \varphi_2, \bigcirc(\varphi_1 \mathcal{U} \varphi_2) \in CL_\varphi$
- $\bullet \varphi \in CL_\varphi$ implies $\varphi \in CL_\varphi$
- $\square \varphi \in CL_\varphi$ implies $\varphi, \bullet \square \varphi \in CL_\varphi$
- $\varphi_1 \mathcal{R} \varphi_2 \in CL_\varphi$ implies $\varphi_1, \varphi_2, \bullet(\varphi_1 \mathcal{R} \varphi_2) \in CL_\varphi$

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

AFW \mathcal{A}_φ associated with an LTL_f formula φ (in NNF)

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

- $2^{\mathcal{P}}$ is the alphabet,
- CL_φ is the state set,
- φ is the initial state
- δ is the transition function, defined as:

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

LTL_f (and LDL_f) formulas can be directly translated in **exponential time** to NFAs, using AFW only implicitly.

NFA \mathcal{A}_φ associated with an LTL_f formula φ (in NNF)

Auxiliary rules

$$\begin{aligned}
 \delta("A", \Pi) &= \text{true if } A \in \Pi \\
 \delta("A", \Pi) &= \text{false if } A \notin \Pi \\
 \delta("¬A", \Pi) &= \text{false if } A \in \Pi \\
 \delta("¬A", \Pi) &= \text{true if } A \notin \Pi \\
 \delta("\varphi_1 \wedge \varphi_2", \Pi) &= \delta("\varphi_1", \Pi) \wedge \delta("\varphi_2", \Pi) \\
 \delta("\varphi_1 \vee \varphi_2", \Pi) &= \delta("\varphi_1", \Pi) \vee \delta("\varphi_2", \Pi) \\
 \delta("○\varphi", \Pi) &= \begin{cases} "\varphi" & \text{if } \text{Last} \notin \Pi \\ \text{false} & \text{if } \text{Last} \in \Pi \end{cases} \\
 \delta("◇\varphi", \Pi) &= \delta("\varphi", \Pi) \vee \delta("○◇\varphi", \Pi) \\
 \delta("\varphi_1 \mathcal{U} \varphi_2", \Pi) &= \delta("\varphi_2", \Pi) \vee (\delta("\varphi_1", \Pi) \wedge \delta("○(\varphi_1 \mathcal{U} \varphi_2)", \Pi)) \\
 \delta("●\varphi", \Pi) &= \begin{cases} "\varphi" & \text{if } \text{Last} \notin \Pi \\ \text{true} & \text{if } \text{Last} \in \Pi \end{cases} \\
 \delta("□\varphi", \Pi) &= \delta("\varphi", \Pi) \wedge \delta("●□\varphi", \Pi) \\
 \delta("\varphi_1 \mathcal{R} \varphi_2", \Pi) &= \delta("\varphi_2", \Pi) \wedge (\delta("\varphi_1", \Pi) \vee \delta("●(\varphi_1 \mathcal{R} \varphi_2)", \Pi))
 \end{aligned}$$

Observe these are the rules defining the transition function of the AFW!

Algorithm

```

algorithm LTLf2NFA
input LTLf formula  $\varphi$ 
output NFA  $\mathcal{A}_\varphi = (2^{\mathcal{P}}, \mathcal{S}, \{s_0\}, \varrho, \{s_f\})$ 
 $s_0 \leftarrow \{"\varphi"\}$  ▷ single initial state
 $s_f \leftarrow \emptyset$  ▷ single final state
 $\mathcal{S} \leftarrow \{s_0, s_f\}, \varrho \leftarrow \emptyset$ 
while ( $\mathcal{S}$  or  $\varrho$  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 true. In building the NFA we assume to have a special proposition $Last \in \mathcal{P}$.

Removing the special proposition $Last$

If we want to remove such an assumption, we can easily transform the obtained NFA

$$A_\varphi = (2^{\mathcal{P}}, S, \{\varphi\}, \varrho, \{\emptyset\}) \text{ into the new NFA } A'_\varphi = (2^{\mathcal{P}'}, S', S_0, \varrho', F')$$

where:

- $\mathcal{P}' = \mathcal{P} - \{Last\}$;
- $S'_0 = \{s_0\}$;
- $S' = S \cup \{ended\}$;
- $F' = \{\emptyset, ended\}$;
- $(q, \Pi', q') \in \varrho'$ iff $\begin{cases} (q, \Pi', q') \in \varrho \text{ or} \\ (q, \Pi' \cup \{Last\}, \emptyset) \in \varrho \text{ and } q' = ended \end{cases}$

Example (NFA for $\Box A$)

The NFA for $\Box A$ is as follows:

- Initial state $\{\Box A\}$;
- Final state $\{\emptyset\}$;
- Transitions:
 - $\rho_n(\{\Box A\}, A \wedge Last, q')$ with $q' \models \delta(\Box A, A \wedge Last) = \delta(A, A \wedge Last) \wedge \delta(\bullet \Box A, A \wedge Last) = \text{true} \wedge \delta(\bullet \Box A, A \wedge Last)$, i.e., $q' = \{\emptyset\}$;
 - $\rho_n(\{\Box A\}, A \wedge \neg Last, q')$ with $q' \models \delta(\Box A, A \wedge \neg Last) = \delta(A, A \wedge \neg Last) \wedge \delta(\bullet \Box A, A \wedge \neg Last) = \text{true} \wedge \delta(\bullet \Box A, A \wedge \neg Last)$, i.e., $q' = \{\Box A\}$;
 - $\rho_n(\{\Box A\}, \neg A, q')$ with $q' \models \delta(\Box A, \neg A) = \delta(A, \neg A) \wedge \delta(\bullet \Box A, \neg A) = \text{false} \wedge \delta(\bullet \Box A, \neg A)$, i.e., there are not such q' . (Notice same behavior with $Last$ and $\neg Last$.)

Example (NFA for $\diamond A$)

The NFA for $\diamond A$ is as follows:

- Initial state $\{\diamond A\}$;
- Final state $\{\emptyset\}$;
- Transitions:
 - $\rho_n(\{\diamond A\}, A \wedge Last, q')$ with $q' \models \delta(\diamond A, A \wedge Last) = \delta(A, A \wedge Last) \vee \delta(\circ \diamond A, A \wedge Last) = \text{true} \vee \text{false}$, i.e., $q' = \{\emptyset\}$;
 - $\rho_n(\{\diamond A\}, A \wedge \neg Last, q')$ with $q' \models \delta(\diamond A, A \wedge \neg Last) = \delta(A, A \wedge \neg Last) \vee \delta(\circ \diamond A, A \wedge \neg Last) = \text{true} \vee \diamond A$, i.e., $q' = \{\emptyset\}$;
 - $\rho_n(\{\diamond A\}, \neg A \wedge Last, q')$ with $q' \models \delta(\diamond A, \neg A \wedge Last) = \delta(A, \neg A \wedge Last) \vee \delta(\circ \diamond A, \neg A \wedge Last) = \text{false} \vee \text{false}$, i.e., no such q' exists;
 - $\rho_n(\{\diamond A\}, \neg A \wedge \neg Last, q')$ with $q' \models \delta(\diamond A, \neg A \wedge \neg Last) = \delta(A, \neg A \wedge \neg Last) \vee \delta(\circ \diamond A, \neg A \wedge \neg Last) = \text{false} \vee \delta(\circ \diamond A, \neg A \wedge \neg Last)$, i.e., $q' = \{\diamond A\}$.

Example (NFA for $\Box\Diamond a$)

The NFA for $\Box\Diamond a$ is as follows:

- Initial state $\{\Box\Diamond a\}$;
- Final state $\{\emptyset\}$;
- Other states $\{\Diamond a, \Box\Diamond a\}$;
- Transitions:
 - $\rho_N(\{\Box\Diamond a\}, a \wedge Last, q')$ with
 $q' \models \delta(\Box\Diamond a, a \wedge Last) = \delta(\Diamond a, a \wedge Last) \wedge \delta(\bullet\Box\Diamond a, a \wedge Last) = \delta(a, a \wedge Last) \vee \delta(\Box\Diamond a, a \wedge Last) = \delta(a, a \wedge Last)$, i.e., $q' = \{\emptyset\}$;
 - $\rho_N(\{\Box\Diamond a\}, a \wedge \neg Last, q')$ with
 $q' \models \delta(\Box\Diamond a, a \wedge \neg Last) = \delta(\Diamond a, a \wedge \neg Last) \wedge \delta(\bullet\Box\Diamond a, a \wedge \neg Last) = (\delta(a, a \wedge \neg Last) \vee \delta(\Box\Diamond a, a \wedge \neg Last)) \wedge \Box\Diamond a$, i.e., $q' = \{\Box\Diamond a\}$;
 - $\rho_N(\{\Box\Diamond a\}, \neg a \wedge Last, q')$ with
 $q' \models \delta(\Box\Diamond a, \neg a \wedge Last) = \delta(\Diamond a, \neg a \wedge Last) \wedge \delta(\bullet\Box\Diamond a, \neg a \wedge Last) = \delta(\Diamond a, \neg a \wedge Last) = \delta(a, \neg a \wedge Last) \vee \delta(\Box\Diamond a, \neg a \wedge Last) = \text{false}$,
 i.e., there exists no such q' ;
 - $\rho_N(\{\Box\Diamond a\}, \neg a \wedge \neg Last, q')$ with
 $q' \models \delta(\Box\Diamond a, \neg a \wedge \neg Last) = \delta(\Diamond a, \neg a \wedge \neg Last) \wedge \delta(\bullet\Box\Diamond a, \neg a \wedge \neg Last) = (\delta(a, \neg a \wedge \neg Last) \vee \Diamond a) \wedge \Box\Diamond a$, i.e., $q' = \{\Diamond a, \Box\Diamond a\}$;
 - $\rho_N(\{\Diamond a, \Box\Diamond a\}, a \wedge Last) = \delta(\Diamond a, a \wedge Last) \wedge \delta(\Box\Diamond a, a \wedge Last)$; this gives rise to: $q' = \{\emptyset\}$;
 - $\rho_N(\{\Diamond a, \Box\Diamond a\}, a \wedge \neg Last) = \delta(\Diamond a, a \wedge \neg Last) \wedge \delta(\Box\Diamond a, a \wedge \neg Last) = \delta(\Diamond a, a \wedge \neg Last) \wedge \delta(\Diamond a, a \wedge \neg Last) \wedge \delta(\bullet\Box\Diamond a, a \wedge \neg Last)$; this gives rise to: $q' = \{\Box\Diamond a\}$;
 - $\rho_N(\{\Diamond a, \Box\Diamond a\}, \neg a \wedge Last) = \delta(\Diamond a, \neg a \wedge Last) \wedge \delta(\Box\Diamond a, \neg a \wedge Last) = \text{false}$, i.e., there exists no such q' ;
 - $\rho_N(\{\Diamond a, \Box\Diamond a\}, \neg a \wedge \neg Last) = \delta(\Diamond a, \neg a \wedge \neg Last) \wedge \delta(\Box\Diamond a, \neg a \wedge \neg Last) = \delta(\Diamond a, \neg a \wedge \neg Last) \wedge \delta(\Diamond a, \neg a \wedge \neg Last) \wedge \delta(\bullet\Box\Diamond a, \neg a \wedge \neg Last)$; this gives rise to: $q' = \{\Diamond a, \Box\Diamond a\}$.

To define the AFW \mathcal{A}_φ associated with an LDL_f formula φ (in NNF), we need first to introduce its syntactic closure.

Syntactic Closure of an LDL_f formula

The syntactic closure, also called Fisher-Ladner closure, CL_φ of an LDL_f formula φ is a set of LDL_f formulas inductively defined as follows:

- $\varphi \in CL_\varphi$
- $\neg\psi \in CL_\varphi$ if $\psi \in CL_\varphi$ and ψ not of the form $\neg\psi'$
- $\varphi_1 \wedge \varphi_2 \in CL_\varphi$ implies $\varphi_1, \varphi_2 \in CL_\varphi$
- $\langle \rho \rangle \varphi \in CL_\varphi$ implies $\varphi \in CL_\varphi$
- $\langle \phi \rangle \varphi \in CL_\varphi$ implies $\phi \in CL_\varphi$ (ϕ is propositional)
- $\langle \psi? \rangle \varphi \in CL_\varphi$ implies $\psi \in CL_\varphi$
- $\langle \rho_1; \rho_2 \rangle \varphi \in CL_\varphi$ implies $\langle \rho_1 \rangle \langle \rho_2 \rangle \varphi \in CL_\varphi$
- $\langle \rho_1 + \rho_2 \rangle \varphi \in CL_\varphi$ implies $\langle \rho_1 \rangle \varphi, \langle \rho_2 \rangle \varphi \in CL_\varphi$
- $\langle \rho^* \rangle \varphi \in CL_\varphi$ implies $\langle \rho \rangle \langle \rho^* \rangle \varphi \in CL_\varphi$

We then put all formulas in NNF. Observe that the cardinality of CL_φ is linear in the size of φ .

AFW \mathcal{A}_φ associated with an LDL_f formula φ (in NNF)

$\mathcal{A}_\varphi = (2^{\mathcal{P}}, CL_\varphi, " \varphi ", \delta, \{ \})$ where, as before, $2^{\mathcal{P}}$ is the alphabet; CL_φ is the state set, φ is the initial state; and δ is the transition function, defined as:

$\delta(\text{tt}, \Pi)$	=	true
$\delta(\text{ff}, \Pi)$	=	false
$\delta(\varphi_1 \wedge \varphi_2, \Pi)$	=	$\delta(\varphi_1, \Pi) \wedge \delta(\varphi_2, \Pi)$
$\delta(\varphi_1 \vee \varphi_2, \Pi)$	=	$\delta(\varphi_1, \Pi) \vee \delta(\varphi_2, \Pi)$
$\delta(\langle \phi \rangle \varphi, \Pi)$	=	$\begin{cases} \text{false} & \text{if } \Pi \not\models \phi \text{ or } \Pi = \epsilon \text{ (trace ended)} \\ \mathbf{e}(\varphi) & \text{o/w } (\phi \text{ propositional}) \end{cases}$
$\delta(\langle \psi? \rangle \varphi, \Pi)$	=	$\delta(\langle \psi \rangle \varphi, \Pi) \wedge \delta(\varphi, \Pi)$
$\delta(\langle \rho_1 + \rho_2 \rangle \varphi, \Pi)$	=	$\delta(\langle \rho_1 \rangle \varphi, \Pi) \vee \delta(\langle \rho_2 \rangle \varphi, \Pi)$
$\delta(\langle \rho_1; \rho_2 \rangle \varphi, \Pi)$	=	$\delta(\langle \rho_1 \rangle \langle \rho_2 \rangle \varphi, \Pi)$
$\delta(\langle \rho^* \rangle \varphi, \Pi)$	=	$\delta(\varphi, \Pi) \vee \delta(\langle \rho \rangle \mathbf{f}_{\langle \rho^* \rangle} \varphi, \Pi)$
$\delta([\phi] \varphi, \Pi)$	=	$\begin{cases} \text{true} & \text{if } \Pi \not\models \phi \text{ or } \Pi = \epsilon \text{ (trace ended)} \\ \mathbf{e}(\varphi) & \text{o/w } (\phi \text{ propositional}) \end{cases}$
$\delta([\psi?] \varphi, \Pi)$	=	$\delta(\text{nnf}(\neg \psi), \Pi) \vee \delta(\varphi, \Pi)$
$\delta([\rho_1 + \rho_2] \varphi, \Pi)$	=	$\delta([\rho_1] \varphi, \Pi) \wedge \delta([\rho_2] \varphi, \Pi)$
$\delta([\rho_1; \rho_2] \varphi, \Pi)$	=	$\delta([\rho_1][\rho_2] \varphi, \Pi)$
$\delta([\rho^*] \varphi, \Pi)$	=	$\delta(\varphi, \Pi) \wedge \delta([\rho] \mathbf{t}_{[\rho^*]} \varphi, \Pi)$
$\delta(\mathbf{f}_\psi, \Pi)$	=	false
$\delta(\mathbf{t}_\psi, \Pi)$	=	true

($\mathbf{e}(\varphi)$ replaces in φ all occurrences of \mathbf{t}_ψ and \mathbf{f}_ψ by $\mathbf{e}(\psi)$)

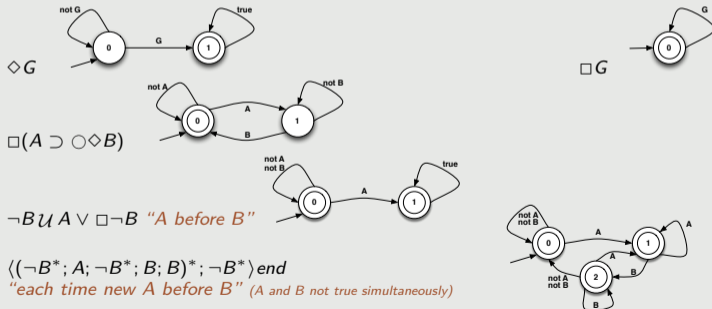
Key point

LTL_f/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.

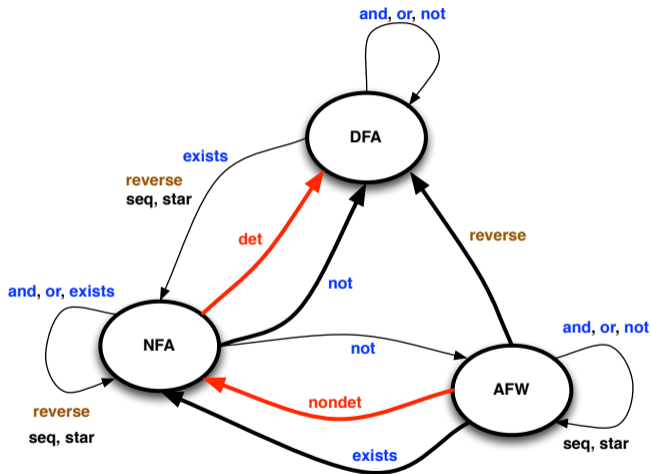
Example (Automata for some LTL_f/LDL_f formulas)



(online software for LTL_f2DFA: <http://ltlf2dfa.diag.uniroma1.it>)

(online software for LDL_f2DFA: <https://flloot.herokuapp.com/>)

Summary of automata theory on finite sequences:



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

LTL_f/LDL_f Satisfiability (φ SAT)

- 1: Given LTL_f/LDL_f formula φ
- 2: Compute AFW for φ (*linear*)
- 3: Compute corresponding NFA (*exponential*)
- 4: Check NFA for nonemptiness (*NLOGSPACE*)
- 5: Return result of check

LTL_f/LDL_f Validity (φ VAL)

- 1: Given LTL_f/LDL_f formula φ
- 2: Compute AFW for $\neg\varphi$ (*linear*)
- 3: Compute corresponding NFA (*exponential*)
- 4: Check NFA for nonemptiness (*NLOGSPACE*)
- 5: Return complemented result of check

LTL_f/LDL_f Logical Implication ($\Gamma \models \varphi$)

- 1: Given LTL_f/LDL_f formulas Γ and φ
- 2: Compute AFW 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. (Construction of NFA can be done while checking nonemptiness.)

As for the infinite traces.

LTL_f/LDL_f Verification

Given a **transition system** \mathcal{T} (i.e. a planning domain or a process/behavior), check that all executions allowed by \mathcal{T} satisfy an **LTL_f/LDL_f specification** φ .

Key Observation

\mathcal{T} can be seen as an automaton by considering every state of \mathcal{T} as accepting.

Hence, we have the following verification algorithm:

- 1: Given Transition System \mathcal{T} and LTL_f/LDL_f formula φ
- 2: Compute AFW for $\neg\varphi$ (*linear in φ*)
- 3: Compute corresponding NFA \mathcal{A} (*exponential in φ*)
- 4: Compute NFA \mathcal{AT} for (\mathcal{T} AND \mathcal{A}) (*polynomial*)
- 4: Check resulting NFA \mathcal{AT} for nonemptiness (*NLOGSPACE*)
- 5: Return complemented result of check

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

As for the infinite traces.

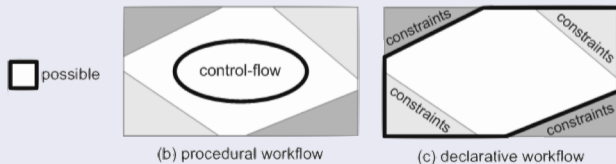
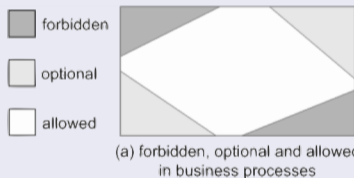
- 1 Motivation
- 2 LTL_f : LTL on Finite Traces
- 3 LTL_f : Expressive Power
- 4 LDL_f : Linear Dynamic Logic on Finite Traces
- 5 LTL_f/ LDL_f Reasoning and Verification
- 6 LTL_f/ LDL_f Program Synthesis**
- 7 Conclusion

LTL_f/LDL_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/LDL_f constraints, **extract process/program/domain** description/transition system that captures **exactly** specification.



(From DECLARE [PesciBovsnavkiDraganVanDerAalst10])

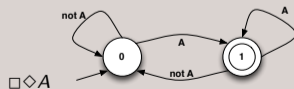
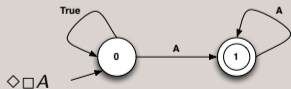
Process corresponding to LTL_f/LDL_f specification always exists for finite traces!

- 1: Given LTL_f/LDL_f formula φ
- 2: Compute AFW for φ (*linear in φ*)
- 3: Compute corresponding NFA (*exponential in φ*)
- 4: Compute corresponding DFA (*exponential in NFA*)
- 5: Trim DFA to avoid dead ends (polynomial)
- 6: Optional: Minimize DFA (polynomial)
- 7: Return resulting DFA

IMPORTANT

This is a BEAUTIFUL RESULT, which does NOT hold in the infinite trace settings!
[\[AbadiLamportWolper89\]](#)

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



Program Synthesis

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

Reactive Synthesis

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

Planning in Fully Observable Nondeterministic domain

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

Planning = game between two players

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

Algorithms

EXPTIME-complete.

But we have **very good** algorithms.

(The entire ICAPS community involved!)

Reactive Synthesis

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

Synthesis = game between two players

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

Algorithms

2EXPTIME-complete.

But we only have **non-scalable** algorithms.

(In spite of 30 years of research!)

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

Solving reactive synthesis

Algorithm for LTL synthesis

Given LTL formula φ

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

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

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

Reactive Synthesis for LTL_f/LDL_f Specifications

- Focus on finite traces [DeGiacomoVardi2013,DeGiacomoVardi2015,DeGiacomoVardi2016]
- Specify **task with LTL_f/LDL_f formulas**
- Rely on transformation of LTL_f/LDL_f formulas into **automata on finite traces** (much more well-behaved wrt infinite traces)
- Same theory as reactive synthesis, but **implementable!**

Reactive synthesis

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

Can the agent set the values of \mathcal{Y} in such a way that for all possible values of \mathcal{X} a certain LTL_f/LDL_f formula remains true?

- **Solution:** compute a function $f : (2^{\mathcal{X}})^* \rightarrow 2^{\mathcal{Y}}$ such that for all generated traces π with X_i arbitrary and $Y_i = f(\pi_{\mathcal{X}}|_i)$, we have that π satisfies the formula ϕ .

Algorithm for LDL_f/LTL_f synthesis

- 1: Given LTL_f/LDL_f formula φ
- 2: Compute AFW for φ (**linear**)
- 2: Compute corresponding NFA (**exponential**)
- 3: Determinize NFA to DFA (**exponential**)
- 4: Synthesize winning strategy for DFA game (**linear**)
- 5: Return strategy

Thm: LTL_f/LDL_f synthesis is 2-EXPTIME-complete.

Same as for infinite traces

DFA games

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

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

Winning condition for DFA games

Let

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

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})
- $Win_{i+1}(\mathcal{G}) = Win_i(\mathcal{G}) \cup PreC(Win_i(\mathcal{G}))$
- $Win(\mathcal{G}) = \bigcup_i Win_i(\mathcal{G})$

From states in $Win(\mathcal{G})$ we can easily extract winning strategies.

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

To **actually compute a strategy**, we define a **strategy generator** based on the winning sets $Win_i(\mathcal{G})$. This is a nondeterministic transducer, where nondeterminism is of the kind **don't-care**: all nondeterministic choices are equally good.

Strategy generator

The strategy generator is a transducer $\mathcal{T}_{\mathcal{G}} = (2^{\mathcal{X} \times \mathcal{Y}}, S, s_0, \varrho, \omega)$ where:

- $2^{\mathcal{X} \times \mathcal{Y}}$ is the alphabet of the transducer;
- S are the states of the transducer;
- s_0 is the initial state;
- $\varrho : S \times 2^{\mathcal{X}} \rightarrow 2^S$ is the transition function such that

$$\varrho(s, X) = \{s' \mid s' = \delta(s, (X, Y)) \text{ and } Y \in \omega(s)\};$$

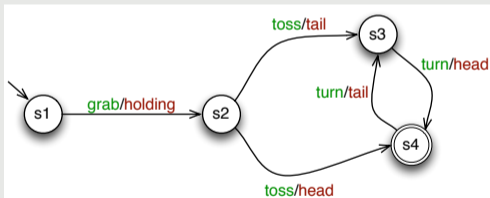
- $\omega : S \rightarrow 2^{\mathcal{Y}}$ is the output function such that

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

The transducer $\mathcal{T}_{\mathcal{G}}$ generates strategies in the following sense: for every way of further restricting $\omega(s)$ to return only one of its values (chosen arbitrarily), we get a strategy.

Example (Toss a coin)

Consider the following (very simple) DFA game. Where the agent can **grab** a coin, **toss** it and **turn** it and the environment responds to grab with the deterministic effect **holding**, to toss by **tail or head** (devilish nondeterminism), and to turn by (deterministically) **changing the coin side**. The goal of the game is to choose appropriately **grab**, **toss**, and **turn** to get **head** in the hand.



Example (Toss a coin)

Compute the winning set

- $Win_0 = \{s_4\}$ (the final states of the game)
- $Win_1 = Win_0 \cup \{s \in S \mid \exists Y \in 2^{\mathcal{Y}}. \forall X \in 2^{\mathcal{X}}. \delta(s, (X, Y)) \in Win_0\} = \{s_4\} \cup \{s_3\}$
- $Win_2 = Win_1 \cup \{s \in S \mid \exists Y \in 2^{\mathcal{Y}}. \forall X \in 2^{\mathcal{X}}. \delta(s, (X, Y)) \in Win_1\} = \{s_3, s_4\} \cup \{s_2\}$
- $Win_3 = Win_2 \cup \{s \in S \mid \exists Y \in 2^{\mathcal{Y}}. \forall X \in 2^{\mathcal{X}}. \delta(s, (X, Y)) \in Win_2\} = \{s_2, s_3, s_4\} \cup \{s_1\}$

So the agent win from all states!

Compute the strategy generator

In fact it is necessary to compute only the output function ω (the rest of the trasducer is determined by such an ω):

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

In our case:

$$\begin{aligned}\omega(s_1) &= \{grab\} \\ \omega(s_2) &= \{toss\} \\ \omega(s_3) &= \{turn\} \\ \omega(s_4) &= \{\} \quad \textit{it is the goal state!}\end{aligned}$$

- 1 Motivation
- 2 LTL_f : LTL on Finite Traces
- 3 LTL_f : Expressive Power
- 4 LDL_f : Linear Dynamic Logic on Finite Traces
- 5 LTL_f/LDL_f Reasoning and Verification
- 6 LTL_f/LDL_f Program Synthesis
- 7 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 only lately clean solutions have been devised.
- We have surveyed results on expressing temporal constraints/goals on finite traces, by reconstructing and integrating results coming from some classical papers.
- We have seen that standard LTL_f on finite traces is less expressive than expected, and that we can extend its expressiveness at no cost.
- We have presented an example of logic that has the same naturalness of LTL_f but the “right” expressive power: LDL_f (a nice combination of LTL_f and RE).
- We have looked at three basic tasks:
 - Satisfiability (Validity, Logical Implication)
 - Verification
 - Synthesis

- Interpreting temporal constraints/goals on finite traces is different than interpreting them on infinite traces (and much more well-behaved)
- When expressing temporal constraints and temporally extend goals we can add to usual LTL_f more powerful constructs a la LDL_f at no cost (possibly for future versions of PDDL).
- There are very general and effective techniques for reasoning, verification and synthesis in this setting – it's not just theory.
- In perspective, the Planning community may come up with a new generation of performing algorithms to deal with these basic tasks (after all, these are all compilable to reachability in large search spaces).