Linear Temporal Logics on Finite Traces: LTL_f and LDL_f

Giuseppe De Giacomo

Università degli Studi di Roma “La Sapienza”
Roma, Italy

Outline

1. Motivation
2. LTL_f: LTL on Finite Traces
3. LTL_f: Expressive Power
4. LDL_f: Linear Dynamic Logic on Finite Traces
5. $\mathrm{LTL}_f/\mathrm{LDL}_f$ Reasoning and Verification
6. $\mathrm{LTL}_f/\mathrm{LDL}_f$ Program Synthesis
7. Conclusion

Outline

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

Motivation: AI

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 [GereviniHslumLongSaettiDimopoulos09 - 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]

Motivation: AI

Temporal extended goals and constraints in AI

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

However:

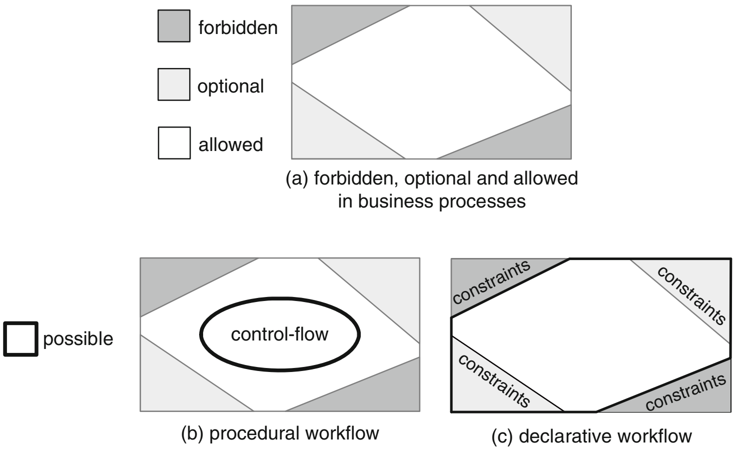
- Often, LTL is interpreted on finite trajectories/traces.
- Often, distinction between interpreting LTL on infinite or on finite traces is blurred.

- 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]

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

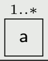
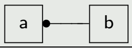
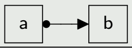
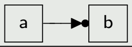
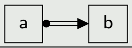
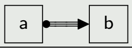
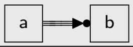
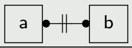
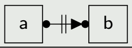
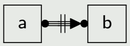
(b) procedural workflow

(c) declarative workflow

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		$\Box(a \supset \Diamond b)$	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		$\Box(a \supset \bigcirc(\neg a \mathcal{U} b))$	Every a must be followed by b, without any other a inbetween
Chain Response		$\Box(a \supset \bigcirc b)$	If a is executed then b must be executed next
Chain Precedence		$\Box(\bigcirc b \supset a)$	Task b can be executed only immediately after a
Not Coexistence		$\neg(\Diamond a \land \Diamond b)$	Only one among tasks a and b can be executed
Neg. Succession		$\Box(a \supset \neg \Diamond b)$	Task a cannot be followed by b, and b cannot be preceded by a
Neg. Chain Succ.		$\Box(a \supset \bigcirc \neg b)$	Tasks a and b cannot be executed next to each other

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

Outline

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 over finite traces

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)*
- $\Diamond\varphi \doteq \mathtt{true}\,\mathcal{U}\,\varphi$: "$\varphi$ will eventually hold"
- $\Box\varphi \doteq \neg\Diamond\neg\varphi$: "from current till last instant φ will always hold"
- $Last \doteq \neg\bigcirc\mathtt{true}$: denotes last instant of trace.

Main formal properties:

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

LTL over finite traces

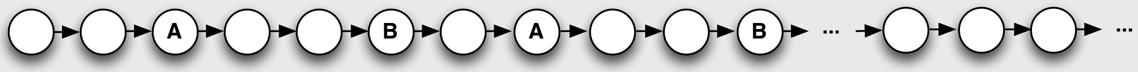
Assuming finite or infinite traces has big impact.

Example

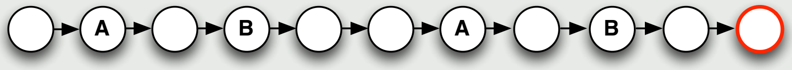
Consider the following formula:

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

- On infinite traces:



- On finite traces:



LTL over finite traces

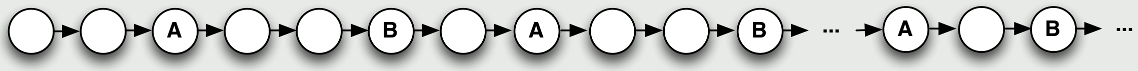
Interpreting LTL on infinite or finite traces has big impact.

Example

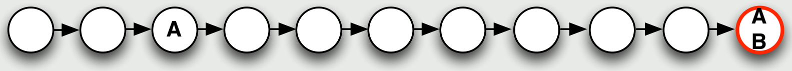
Consider the following formula:

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

- On infinite traces:



- On finite traces:



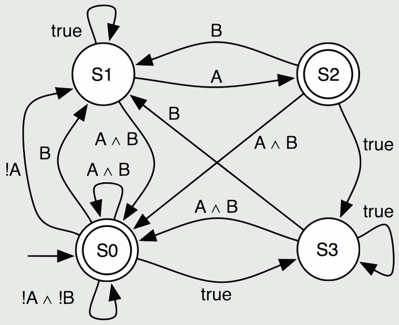
LTL over finite traces

Interpreting LTL on infinite or finite traces has big impact.

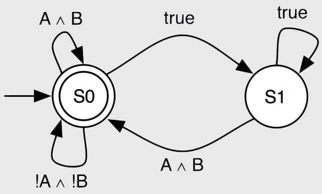
Example

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:



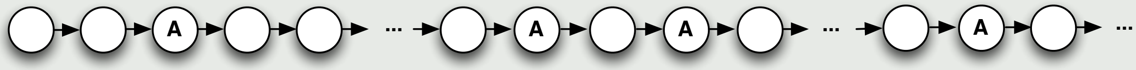
LTL over finite traces

Example (Unintuitive LTL$_f$ formulas - "Response")

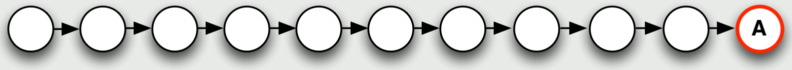
$$\Box\Diamond A$$

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. $\Diamond(Last \wedge A)$



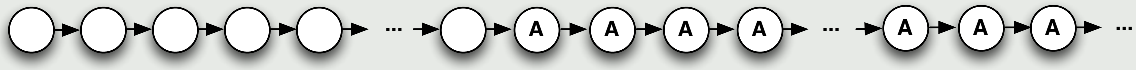
LTL over finite traces

Example (Unintuitive LTL$_f$ formulas - "Persistence")

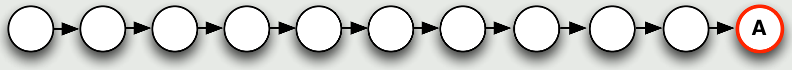
$$\Diamond\Box\varphi$$

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.*

LTL over finite traces

Example (Unintuitive LTL$_f$ formulas)

- $\Box\Diamond\varphi$: 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.:

  $$\Diamond(Last \wedge \varphi).$$

  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.

- $\Diamond\Box\varphi$: 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.:

  $$\Diamond(Last \wedge \varphi).$$

*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.*

LTL over finite 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;
  - $\Box(\bigcirc A \supset \bigwedge_{F\not\in Add(A)\cup Del(A)}(F \equiv \bigcirc F))$: 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: $\bigwedge_{F\in Init} F \wedge \bigwedge_{F\not\in Init} \neg F$.
- Finally goal φ_g eventually holds: $\Diamond\varphi_g$.

Thm: A plan exists iff the LTL$_f$ formula is SAT.

LTL over finite traces

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:

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

- 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.

Outline

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

Expressive Power of LTL_f

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

FOL over finite sequences (aka $\mathrm{FO}[<]$)

- 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:
  - $succ(x, y) \doteq (x < y) \land \neg\exists z.x < z < y$: 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 $\neg\exists y.succ(x, y)$.

Expressive Power of LTL_f

LTL_f to FOL

We can translate any LTL_f formula to FOL

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

Example

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

And viceversa!

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

LTL_f *has the same expressive power of* FOL.

MSO on Finite Sequences

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.

Regular Expressions as Temporal Properties

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.

Regular Expressions as Temporal Properties

Example

- “Safety” ($\Box\varphi$):

  $$\varphi^*$$

  that means that always, until the end of the trace, φ holds.
- “Liveness” ($\Diamond\varphi$):

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

  that means that eventually before the end of the trace φ holds.
- “Conditional response” ($\Box(\psi \supset \Diamond\varphi)$):

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

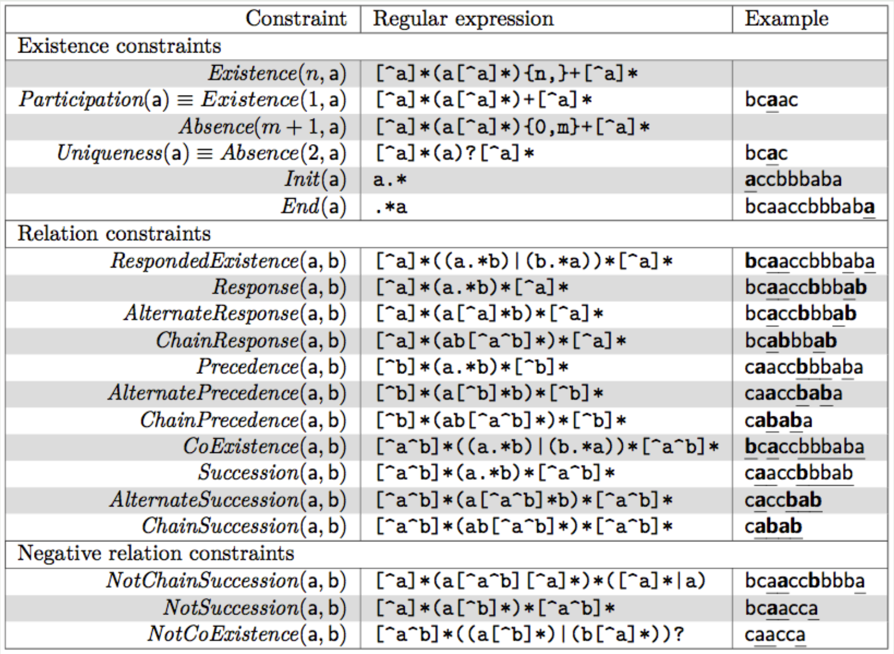
  that means whenever ψ holds then later φ holds.
- “Ordered occurrence”:

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

  that says φ_1 and φ_2 will both happen in order.

Regular Expressions as Temporal Properties

Example (Interesting RE expressions in BPM [DiCiccioMecella12])



Constraint	Regular expression	Example
Existence constraints		
Existence(n, a)	`[^a]*(a[^a]*){n,}+[^a]*`	
Participation(a) ≡ *Existence*(1, a)	`[^a]*(a[^a]*)+[^a]*`	bcaac
Absence(m + 1, a)	`[^a]*(a[^a]*){0,m}+[^a]*`	
Uniqueness(a) ≡ *Absence*(2, a)	`[^a]*(a)?[^a]*`	bcac
Init(a)	`a.*`	accbbbaba
End(a)	`.*a`	bcaaccbbbaba
Relation constraints		
RespondedExistence(a, b)	`[^a]*((a.*b)|(b.*a))*[^a]*`	bcaaccbbbaba
Response(a, b)	`[^a]*(a.*b)*[^a]*`	bcaaccbbbab
AlternateResponse(a, b)	`[^a]*(a[^a]*b)*[^a]*`	bcaccbbbab
ChainResponse(a, b)	`[^a]*(ab[^a^b]*)*[^a]*`	bcabbbab
Precedence(a, b)	`[^b]*(a.*b)*[^b]*`	caaccbbbaba
AlternatePrecedence(a, b)	`[^b]*(a[^b]*b)*[^b]*`	caaccbaba
ChainPrecedence(a, b)	`[^b]*(ab[^a^b]*)*[^b]*`	cababa
CoExistence(a, b)	`[^a^b]*((a.*b)|(b.*a))*[^a^b]*`	bcaccbbbaba
Succession(a, b)	`[^a^b]*(a.*b)*[^a^b]*`	caaccbbbab
AlternateSuccession(a, b)	`[^a^b]*(a[^a^b]*b)*[^a^b]*`	caccbab
ChainSuccession(a, b)	`[^a^b]*(ab[^a^b]*)*[^a^b]*`	cabab
Negative relation constraints		
NotChainSuccession(a, b)	`[^a]*(a[^a^b][^a]*)*([^a]*|a)`	bcaaccbbbba
NotSuccession(a, b)	`[^a]*(a[^b]*)*[^a^b]*`	bcaacca
NotCoExistence(a, b)	`[^a^b]*((a[^b]*)|(b[^a]*))?`	caacca

(Mostly RE translation of DECLARE LTL$_f$ patterns)

Star-free Regular Expressions

Theorem ([McNaughtonPapert1971])

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

Star-free regular expressions

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

where $\overline{\varrho}$ stands for the complement of ϱ, i.e., $\mathcal{L}(\overline{\varrho}) = (2^{\mathcal{P}})^*/\mathcal{L}(\varrho)$.
Star-free regular expressions are strictly less espressive then 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)

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

LTL$_f$ is Equivalent to Star-Free RE

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 $\texttt{true}^*;\phi;\texttt{true}^*$
- $\Box\phi$ can be expressed as $\overline{(\texttt{true}^*;\neg\phi;\texttt{true}^*)}$
- $\phi_1\,\mathcal{U}\,\phi_2$ can be expressed as $\overline{(\texttt{true}^*;\neg\phi_1;\texttt{true}^*)};\phi_2;\texttt{true}^*$

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

Outline

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:

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

- 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).

LDL$_f$: Linear Dynamic Logic on Finite Traces

LTL$_f$ can be translated into LDL$_f$ in linear time

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

RE can be translated into LDL$_f$ in linear time

$$g(\varrho) = \langle\varrho\rangle end$$

where end stands for "the traces ends", i.e., $[\texttt{true}]\texttt{ff}$.

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.*

LDL$_f$: Linear Dynamic Logic 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

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

- $\pi x.\delta(x)$ stands for $\Sigma_{o \in Obj}\ \delta(o)$
- **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.)

LTL$_f$/LDL$_f$: Linear Temporal Logics on Finite Traces

Example

- “*All coffee requests from person p will eventually be served*”:
  $$\Box(request_p \supset \Diamond coffee_p) \qquad [\texttt{true}^*](request_p \supset \langle\texttt{true}^*\rangle coffee_p)$$
- “*Every time the robot opens door d it closes it immediately after*”:
  $$\Box(openDoor_d \supset \bigcirc closeDoor_d) \qquad [\texttt{true}^*]([openDoor_d]closeDoor_d)$$
- “*Before entering restricted area a the robot must have permission for a*”:
  $$\neg inArea_a\, \mathcal{U}\, getPerm_a \vee \Box\neg inArea_a \qquad \langle(\neg inArea_a)^*\rangle getPerm_a \vee [\texttt{true}^*]\neg inArea_a$$
- “*Each time the robot enters the restricted area a it must have a new permission for a*”:
  $$\langle(\neg inArea_a{}^*; getPerm_a; \neg inArea_a{}^*; inArea_a; inArea_a^*)^*; \neg inArea_a{}^*\rangle end$$
- “*At every point, if it is hot then, if the air-conditioning system is off, turn it on, else don’t turn it off*”:
  $$\begin{array}{l} [\texttt{true}^*]\langle\textbf{if } (\mathit{hot}) \textbf{ then} \\ \qquad\qquad \textbf{if } (\neg \mathit{airOn}) \textbf{ then } \mathit{turnOnAir} \\ \qquad\qquad \textbf{else } \neg \mathit{turnOffAir}\rangle\, true \end{array}$$

Outline

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$_f$ and Automata

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 $\mathcal{A}_\varphi$ is the AFW φ is translated into.

We can compile reasoning into automata based procedures!

LTL$_f$ and Automata

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

AFW run

Given an input word $a_0, a_1, \ldots a_{n-1}$, 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 $\mathtt{true}$ 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].)

LTL$_f$ and Automata

To define the AFW $\mathcal{A}_\varphi$ associated with an LTL$_f$ formula φ (in NNF), we need first to introduce its synthactic closure.

Syntactic Closure of an LTL$_f$ formula

The syntactic closure, also called Fisher-Ladner closure, CL_φ of an LTL$_f$ formula φ is a set of LDL$_f$ 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 and Automata

AFW $\mathcal{A}_\varphi$ 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:

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

LTL_f and Automata

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

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

Auxiliary rules

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

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

Algorithm

```
algorithm LTL_f2NFA
input LTL_f formula $\varphi$
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 $\varrho$ 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
```

LTL$_f$ and Automata

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

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

where:

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

LTL$_f$ and Automata: Examples

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

$\textsc{ltl}_f$ and Automata: Examples

Example ($\textsc{nfa}$ for $\Diamond A$)

The $\textsc{nfa}$ for $\Diamond A$ is as follows:

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

$\textsc{ltl}_f$ and Automata: Examples

Example ($\textsc{nfa}$ for $\Box\Diamond a$)

The $\textsc{nfa}$ 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 $q' \models \delta(\Box\Diamond a, a \wedge Last) = \delta(\Diamond a, a \wedge Last) \wedge \delta(\bullet\Box\Diamond a, a \wedge Last) = \delta(a, a \wedge Last) \vee \delta(\bigcirc\Diamond a, a \wedge Last) = \delta(a, a \wedge Last)$, i.e., $q' = \{\emptyset\}$;
  - $\rho_n(\{\Box\Diamond a\}, a \wedge \neg Last, q')$ with $q' \models \delta(\Box\Diamond a, a \wedge \neg Last) = \delta(\Diamond a, a \wedge \neg Last) \wedge \delta(\bullet\Box\Diamond a, a \wedge \neg Last) = (\delta(a, a \wedge \neg Last) \vee \delta(\bigcirc\Diamond a, a \wedge \neg Last)) \wedge \Box\Diamond a$, i.e., $q' = \{\Box\Diamond a\}$;
  - $\rho_n(\{\Box\Diamond a\}, \neg a \wedge Last, q')$ with $q' \models \delta(\Box\Diamond a, \neg a \wedge Last) = \delta(\Diamond a, \neg a \wedge Last) \wedge \delta(\bullet\Box\Diamond a, \neg a \wedge Last) = \delta(\Diamond a, \neg a \wedge Last) =$ $\delta(a, \neg a \wedge Last) \vee \delta(\bigcirc\Diamond a, \neg a \wedge Last) = \mathtt{false}$, 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:$q' = \{\Box\Diamond a\}$;
  - $\rho_n(\{\Diamond a, \Box\Diamond a\}, \neg a \wedge Last) = \delta(\Diamond a, \neg a \wedge Last) \wedge \delta(\Box\Diamond a, a \wedge Last) = \mathtt{false}$, 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: $q' = \{\Diamond a, \Box\Diamond a\}$.

$\textsc{ldl}_f$ and Automata

To define the $\textsc{afw}$ $\mathcal{A}_\varphi$ associated with an $\textsc{ldl}_f$ formula φ (in NNF), we need first to introduce its syntactic closure.

Syntactic Closure of an $\textsc{ldl}_f$ formula

The syntactic closure, also called Fisher-Ladner closure, CL_φ of an $\textsc{ldl}_f$ formula φ is a set of $\textsc{ldl}_f$ formulas inductively defined as follows:

$$\begin{array}{l}
\varphi \in CL_\varphi \\
\neg\psi \in CL_\varphi \text{ if } \psi \in CL_\varphi \text{ and } \psi \text{ not of the form } \neg\psi' \\
\varphi_1 \wedge \varphi_2 \in CL_\varphi \text{ implies } \varphi_1, \varphi_2 \in CL_\varphi \\
\langle\rho\rangle\varphi \in CL_\varphi \text{ implies } \varphi \in CL_\varphi \\
\langle\phi\rangle\varphi \in CL_\varphi \text{ implies } \phi \in CL_\varphi \quad (\phi \text{ is propositional}) \\
\langle\psi?\rangle\varphi \in CL_\varphi \text{ implies } \psi \in CL_\varphi \\
\langle\rho_1;\rho_2\rangle\varphi \in CL_\varphi \text{ implies } \langle\rho_1\rangle\langle\rho_2\rangle\varphi \in CL_\varphi \\
\langle\rho_1+\rho_2\rangle\varphi \in CL_\varphi \text{ implies } \langle\rho_1\rangle\varphi, \langle\rho_2\rangle\varphi \in CL_\varphi \\
\langle\rho^*\rangle\varphi \in CL_\varphi \text{ implies } \langle\rho\rangle\langle\rho^*\rangle\varphi \in CL_\varphi
\end{array}$$

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

$\textsc{ldl}_f$ and Automata

$\textsc{afw}$ $\mathcal{A}_\varphi$ associated with an $\textsc{ldl}_f$ formula φ (in NNF)

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

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

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

$\text{LTL}_f/\text{LDL}_f$ and automata

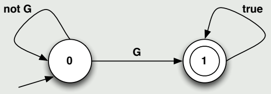
Key point

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

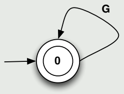
$$t \models \varphi \text{ iff } t \in \mathcal{L}(A_\varphi)$$

where A_φ is the DFA φ is translated into.

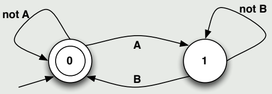
Example (Automata for some $\text{LTL}_f/\text{LDL}_f$ formulas)



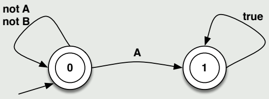
$\Diamond G$



$\Box G$



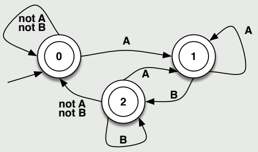
$\Box(A \supset \bigcirc\Diamond B)$



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

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

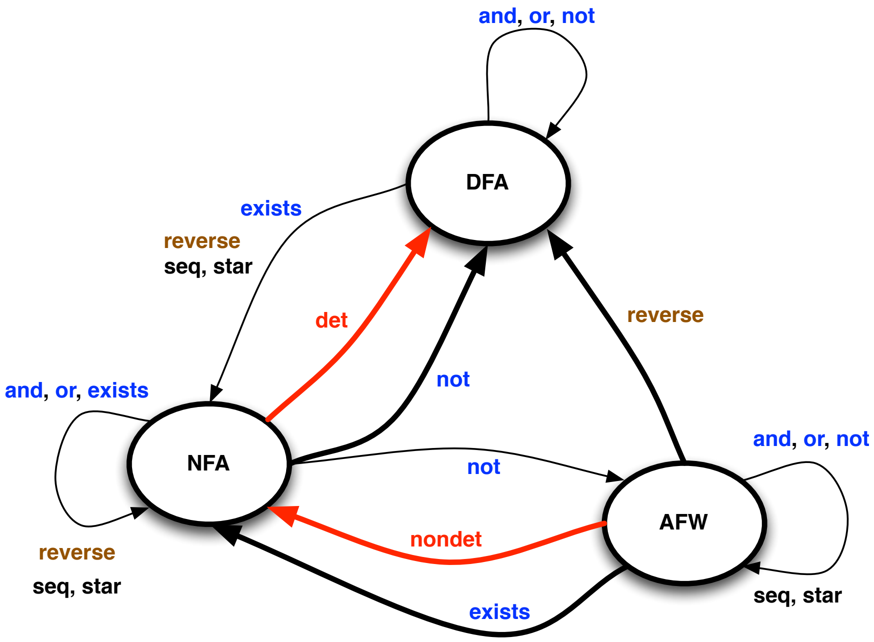
"each time new A before B" (A and B not true simultaneously)



(online software for LTLf2DFA: `http://ltlf2dfa.diag.uniroma1.it`)
(online software for LDLf2DFA: `https://flloat.herokuapp.com`)

$\text{LTL}_f/\text{LDL}_f$ and Automata

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$_f$ Reasoning

LTL$_f$/LDL$_f$ Satisfiability (φ SAT)

```
Given LTL$_f$/LDL$_f$ formula $\varphi$
    Compute AFW for $\varphi$ (linear)
    Compute corresponding NFA (exponential)
    Check NFA for nonemptiness (NLOGSPACE)
Return result of check
```

LTL$_f$/LDL$_f$ Validity (φ VAL)

```
Given LTL$_f$/LDL$_f$ formula $\varphi$
    Compute AFW for $\neg\varphi$ (linear)
    Compute corresponding NFA (exponential)
    Check NFA for nonemptiness (NLOGSPACE)
Return complemented result of check
```

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

```
Given LTL$_f$/LDL$_f$ formulas $\Gamma$ and $\varphi$
    Compute AFW for $\Gamma \wedge \neg\varphi$ (linear)
    Compute corresponding NFA (exponential)
    Check NFA for nonemptiness (NLOGSPACE)
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

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:

```
Given Transition System $\mathcal{T}$ and LTL$_f$/LDL$_f$ formula $\varphi$
    Compute AFW for $\neg\varphi$ (linear in $\varphi$)
    Compute corresponding NFA $\mathcal{A}$ (exponential in $\varphi$)
    Compute NFA $\mathcal{AT}$ for ($\mathcal{T}$ AND $\mathcal{A}$) (polynomial)
    Check resulting NFA $\mathcal{AT}$ for nonemptiness (NLOGSPACE)
Return complemented result of check
```

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

As for the infinite traces.

Outline

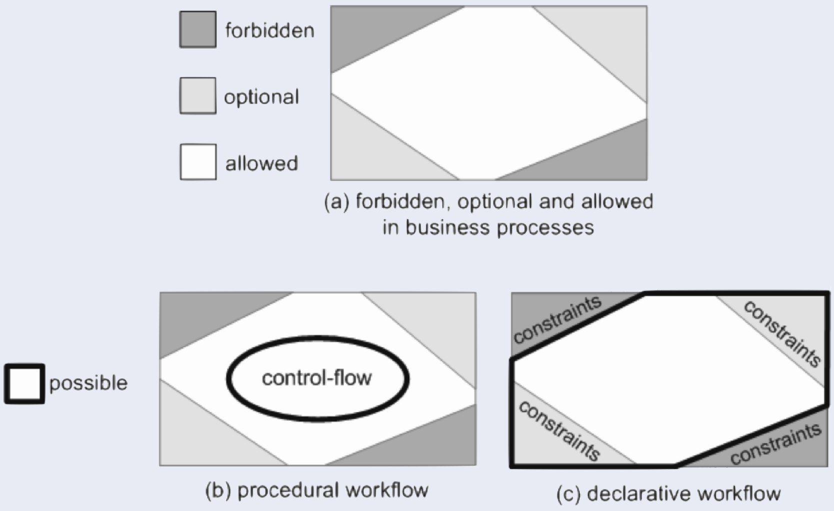
1. Motivation
2. LTL$_f$: LTL on Finite Traces
3. LTL$_f$: Expressive Power
4. LDL$_f$: Linear Dynamic Logic on Finite Traces
5. LTL$_f$/LDL$_f$ Reasoning and Verification
6. LTL$_f$/LDL$_f$ Program Synthesis
7. Conclusion

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.



(a) forbidden, optional and allowed in business processes

(b) procedural workflow

(c) declarative workflow

(From DECLARE [PesicBovsnavkiDraganVanDerAalst10])

LTL$_f$/LDL$_f$ Synthesis Under Full Controllability (BPM)

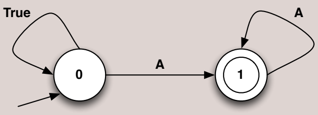
Process corresponding to LTL$_f$/LDL$_f$ specification always exists for finite traces!

```
Given LTL_f/LDL_f formula $\varphi$
    Compute AFW for $\varphi$ (linear in $\varphi$)
    Compute corresponding NFA (exponential in $\varphi$)
    Compute corresponding DFA (exponential in NFA)
    Trim DFA to avoid dead ends (polynomial)
    Optional: Minimize DFA (polynomial)
Return resulting DFA
```

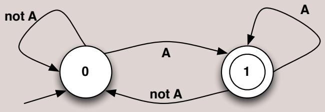
IMPORTANT

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

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



$\Diamond\Box A$



$\Box\Diamond A$

Program Synthesis in Formal Methods

Program Synthesis

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

Reactive Synthesis

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

Planning and Reactive Synthesis

Planning in Fully Observable Nondeterministic domain

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

Planning = game between two players

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

Algorithms

EXPTIME-complete.
But we have very good algorithms.

(The entire ICAPS community involved!)

Reactive Synthesis

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

Synthesis = game between two players

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

Algorithms

2EXPTIME-complete.
But we only have non-scalable algorithms.

(In spite of 30 years of research!)

Focus on finite traces!

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

Solving reactive synthesis

Algorithm for LTL synthesis

```
Given LTL formula $\varphi$
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 Syntesis from $\text{LTL}_f/\text{LDL}_f$ Specifications

Focus on finite traces!

Giuseppe De Giacomo: "We should consider synthesis for finite traces specifications."
Moshe Vardi: "But that is easy."
Giuseppe De Giacomo: "Precisely!" [DeGiacomoVardi2013,DeGiacomoVardi2015,DeGiacomoVardi2016]

Reactive Synthesis from $\text{LTL}_f/\text{LDL}_f$ Specifications

- Specify task with $\text{LTL}_f/\text{LDL}_f$ formulas
- Relay on trasformation of $\text{LTL}_f/\text{LDL}_f$ formulas into automata on finite traces (much more well-behaved wrt infinite traces)
- Follow same theory of reactive synthesis, but now everything is implementable!

Reactive Syntesis from $\text{LTL}_f/\text{LDL}_f$ Specifications

Reactive synthesis

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

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

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

```
Given LTL_f/LDL_f formula $\varphi$
    Compute AFW for $\varphi$ (linear)
    Compute corresponding NFA (exponential)
    Determinize NFA to DFA (exponential)
    Synthesize winning strategy for DFA game (linear)
Return strategy
```

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

Same as for infinite traces

DFA Games

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;
- $\delta : S \times 2^{\mathcal{X}\times\mathcal{Y}} \to S$, transition function of the game: given current state s and a choice of propositions X and Y, respectively for enviroment 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 $\mathcal{G}$.

Computing Strategies

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

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

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

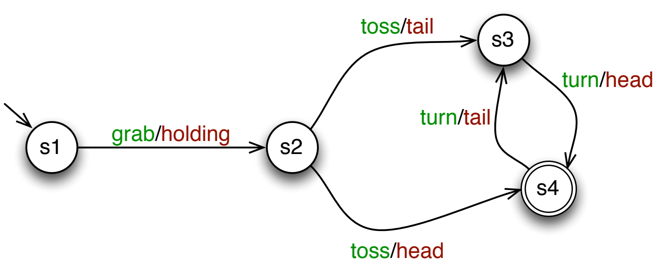
$$\omega(s) = \{Y \mid \text{ if } s \in Win_{i+1}(\mathcal{G}) - Win_i(\mathcal{G}) \text{ then } \forall X.\delta(s, (X, Y)) \in Win_i(\mathcal{G})\}.$$

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 of DFA Game

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 of DFA Game

Example (Toss a coin)

Compute the winning set

- $Win_0 = \{s4\}$ (the final states of the game)
- $Win_1 = Win_0 \cup \{s \in S \mid \exists Y \in 2^{\mathcal{Y}}.\forall X \in 2^{\mathcal{X}}.\delta(s,(X,Y)) \in Win_0\} = \{s4\} \cup \{s3\}$
- $Win_2 = Win_1 \cup \{s \in S \mid \exists Y \in 2^{\mathcal{Y}}.\forall X \in 2^{\mathcal{X}}.\delta(s,(X,Y)) \in Win_1\} = \{s3, s4\} \cup \{s2\}$
- $Win_3 = Win_3 \cup \{s \in S \mid \exists Y \in 2^{\mathcal{Y}}.\forall X \in 2^{\mathcal{X}}.\delta(s,(X,Y)) \in Win_2\} = \{s2, s3, s4\} \cup \{s1\}$

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:

$$\begin{array}{lcl} \omega(s1) & = & \{\mathit{grab}\} \\ \omega(s2) & = & \{\mathit{toss}\} \\ \omega(s3) & = & \{\mathit{turn}\} \\ \omega(s4) & = & \{\} \quad \textit{it is the goal state!} \end{array}$$

Outline

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

Summary

- 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

What to bring home

- 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).