Synthesis in Linear Temporal Logics on Finite Traces

Giuseppe De Giacomo

Outline

Outline

LTL over finite traces

LTL$_f$: the language (in symbols)

Same syntax as standard LTL but interpreted over finite traces

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

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

LTL$_f$: the language (in words)

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

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

LTL over finite traces

In symbols

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

In words

eventually A	"eventually A"	*reachability*
always A	"always A"	*safety*
always$(A \supset$ *eventually* $B)$	"always if A then eventually B"	*reactiveness*
A *until* B	"A until B"	*strong until – stronger than English until*
A *until* $B \vee$ *always* A	"A until B"	*weak until – just like English until*

LTL$_f$ Semantics

Finite Traces

The semantics of LTL$_f$ is given in terms of finite traces denoting a finite sequence of consecutive instants of time.

- Finite traces are finite words π over the alphabet of $2^{\mathcal{P}}$, i.e., as alphabet we have all the possible propositional interpretations of the propositional symbols in $\mathcal{P}$.
- We denote the length of a trace π as $length(\pi)$.
- We denote the positions, i.e. instants, on the trace as π, i with $0 \leq i \leq last$, where $last = length(\pi) - 1$ is the last element of the trace.

LTL_f Semantics

LTL_f Semantics

Given a finite trace π, we inductively define when an LTL_f formula φ is true at an instant i (for $0 \leq i \leq last$), in symbols $\pi, i \models \varphi$, as follows:

- $\pi, i \models A$, for $A \in \mathcal{P}$ iff $A \in \pi(i)$.
- $\pi, i \models \neg\varphi$ iff $\pi, i \not\models \varphi$.
- $\pi, i \models \varphi_1 \wedge \varphi_2$ iff $\pi, i \models \varphi_1$ and $\pi, i \models \varphi_2$.
- $\pi, i \models \bigcirc\varphi$ iff $i{+}1 \leq last$ and $\pi, i{+}1 \models \varphi$.
- $\pi, i \models \bullet\varphi$ iff $i{+}1 \leq last$ implies $\pi, i{+}1 \models \varphi$.
- $\pi, i \models \Diamond\varphi$ iff for some j such that $i \leq j \leq last$, we have $\pi, j \models \varphi$.
- $\pi, i \models \Box\varphi$ iff for all j such that $i \leq j \leq last$, we have $\pi, j \models \varphi$.
- $\pi, i \models \varphi_1 \mathcal{U} \varphi_2$ iff for some j such that $i \leq j \leq last$, we have $\pi, j \models \varphi_2$ and for all k, $i \leq k < j$, we have $\pi, k \models \varphi_1$.

LTL$_f$ Examples

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

$$\Box(request_p \supset \Diamond coffee_p)$$

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

$$\Box(openDoor_d \supset \bigcirc closeDoor_d)$$

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

$$\neg inArea_a \,\mathcal{U}\, getPerm_a \vee \Box \neg inArea_a$$

LTL_f and Automata

Key point

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

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

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

We can compile reasoning into automata based procedures!

Outline

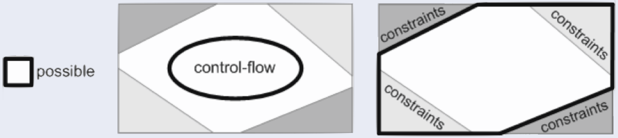
LTL_f Synthesis Under Full Controllability (BPM)

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

Synthesis under full controllability

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

(a) forbidden, optional and allowed in business processes



(b) procedural workflow (c) declarative workflow

(From DECLARE *[PesicBovsnavkiDraganVanDerAalst10])*

LTL_f Synthesis Under Full Controllability (BPM)

Process corresponding to LTL_f specification always exists for finite traces!

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

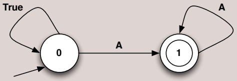
1: Given LTL_f formula φ
2: Compute AFA for φ *(linear in φ)*
3: Compute corresponding NFA *(exponential in φ)*
4: Compute corresponding DFA *(exponential in NFA)*
5: Trim DFA to avoid dead ends (polynomial)
6: Optional: Minimize DFA (polynomial)
7: Return resulting DFA

IMPORTANT

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

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

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



LTL_f Synthesis Under Full Controllability (BPM)

Process corresponding to LTL_f specification always exists for finite traces!

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

1: Given LTL_f formula φ
2: Compute AFA 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

Questions

What happens if

1. we return directly the NFA at step 3? WE CAN'T
2. we return the DFA at step 4, without trimming? WE CAN'T
3. we omit optional step 6? LESS MEMORY USE
4. we perform optional step 6? MORE

Outline

Program Synthesis

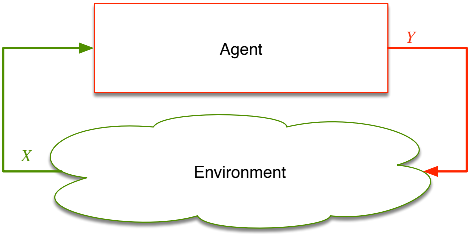
Program Synthesis

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

Reactive Synthesis

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

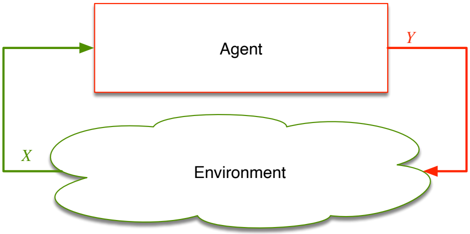
Agent in Environment



Inputs and outputs

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

Synthesis as a Game



Game View

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

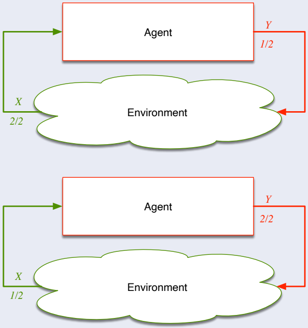
- Agent chooses controllable output $Y \in 2^{\mathcal{Y}}$
- Environment chooses uncontrollable input $X \in 2^{\mathcal{X}}$
- Round: agent and environment set their values
- Play: finite trace τ over $(\mathcal{X} \cup \mathcal{Y})$
- Agent decides when to stop
- Specification: LTL_f formula φ
- Agent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

Synthesis as a Game

Game Rounds

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

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

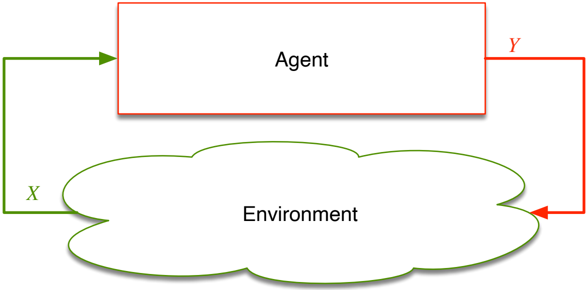


Pair actions and states in a time instant (reminder)

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

- Pair the agent action and the resulting state, (in fact labeling of the state) of the environment $\Longleftarrow$ ACT AND RESP AT THE SAME TIME
  The propositional representation a for an action a will stand for "action a just executed".
- Pair current environment (labeling of the) state and the next action instructed by the agent
  The propositional representation a for an action a will stand for "action a just instructed to be executed next".

Agent strategie



Agent strategies

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

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

where

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

  (a finite sequence of fluents configurations)

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

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

LTL_f Synthesis

Game View

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

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

Realizability and Synthesis

- Agent strategy σ_a is winning if for every infinite sequence $X_0, X_1, X_2 \ldots$ the play $(\sigma_a(\epsilon), X_0)(\sigma_a(X_0), X_1), (\sigma_a(X_0, X_1), X_2) \ldots$ there exists an instant n such that at the trace $\tau = (\sigma_a(\epsilon), X_0), \ldots, (\sigma_a(X_0, \ldots, X_{n-1}), X_n)$ satisfies φ, i.e., $\tau \models \varphi$.
- Realizability: exists a winning agent strategy σ_a.
- Synthesis: obtain a winning agent strategy σ_a.

Synthesis from LTL$_f$ Specifications

Synthesis from LTL$_f$ Specifications

- Specify task with LTL$_f$ formulas
- Relay on trasformation of LTL$_f$ formulas into DFA
- Solve game on the DFA: find an agent strategy to reach a final state in spite of how the environment reacts.

Synthesis from LTL$_f$ Specifications

Synthesis from LTL$_f$ Specifications

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

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

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

Algorithm for LTL$_f$ synthesis

1: Given LTL$_f$ formula φ
2: Compute AFA for φ (linear)
2: Compute corresponding NFA (exponential)
3: Determinize NFA to DFA (exponential)
4: Synthesize winning strategy for DFA game (linear)
5: Return strategy

Thm: LTL$_f$ synthesis is 2-EXPTIME-complete.

Same as for infinite traces

DFA Games

DFA games

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

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

Winning condition for DFA games

Let

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

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

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

$\mu Z. F \vee PREADV([Z])$

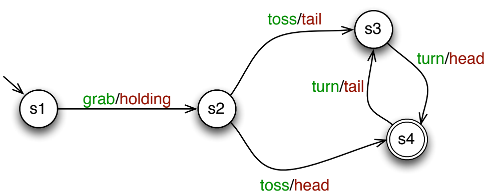
$\mu Z. F \vee \langle \quad \rangle Z$

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

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

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:

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

Computing Strategies

To actually compute a strategy, we need to

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

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

- Compute the corresponding strategy $\sigma_a : (2^{\mathcal{X}})^* \to 2^{\mathcal{Y}}$ via a transducer $\mathcal{T}_G$ obtained form the game $\mathcal{G}$ and the function $choice(\omega(s))$.
  The obtained σ_a is memory-full, but has only finite number of states.

Strategy as a transducer

Given the DFA game $\mathcal{G} = (2^{\mathcal{X}\cup\mathcal{Y}}, S, s_0, \delta, F)$ and the function $choice(\omega(s))$, the strategy returned is a transducer

$$\mathcal{T}_{\mathcal{G}} = (2^{\mathcal{X}}, S, s_0, \varrho, \omega_{choice})$$

where:

- $2^{\mathcal{X}}$ is the alphabet of the transducer;
- S are the states of the transducer;
- s_0 is the initial state;
- $\varrho : S \times 2^{\mathcal{X}} \to S$ is the transition function (partial) such that $\varrho(s,X) = \delta(s,(X,choice(\omega(s))))$
- $\omega_{choice} : S \to 2^{\mathcal{Y}}$ is the output function such that $\omega_{choice} = choice(\omega(s))$

Example

Example

Consider the following LTL_f formula φ:

$$
\begin{aligned}
&(\neg Printed_1 \wedge \neg Printed_2 \wedge Delivered \wedge \\
&\ \Box(\bullet(print \supset \Diamond(Printed_1 \vee Printed_2))) \wedge \\
&\ \Box(Printed_1 \supset \bullet(collect_1 \supset \Diamond Delivered)) \wedge \\
&\ \Box(Printed_2 \supset \bullet(collect_2 \supset \Diamond Delivered))) \\
&\qquad \supset \Diamond Delivered) \wedge \\
&\Box((print \supset \neg collect_1 \wedge \neg collect_2) \wedge (collect_1 \supset \neg print \wedge \neg collect_2) \wedge (collect_2 \supset \neg print \wedge \neg collect_1))
\end{aligned}
$$

where the agent controls $print, collect_1, collect_2$ and the environment controls $Printed_1, Printed_2, Delivered$.

Question: Synthesize a strategy σ_a for realizing φ.

Synthesis in LTL

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

Solving reactive synthesis

Algorithm for LTL synthesis

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

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

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