Reasoning in Linear Temporal Logics on Finite Traces

Giuseppe De Giacomo

Outline

Outline

LTL$_f$: the language (in symbols)

Same syntax as standard LTL but interpreted over finite traces

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

- A: atomic propositions
- $\neg\varphi$, $\varphi_1 \land \varphi_2$, $\varphi_1 \lor \varphi_2$, $\varphi_1 \supset \varphi_2$: 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:

$$\varphi ::= A \mid \quad \neg\varphi \mid \varphi_1 \land \varphi_2 \mid \varphi_1 \lor \varphi_2 \mid \varphi_1 \supset \varphi_2 \mid \quad \mathit{next}\,\varphi \mid \mathit{wnext}\,\varphi \mid \mathit{eventually}\,\varphi \mid \mathit{always}\,\varphi \mid \varphi_1\ \mathit{until}\,\varphi_2$$

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":

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

Outline

LTL_f and Automata

Key point

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

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

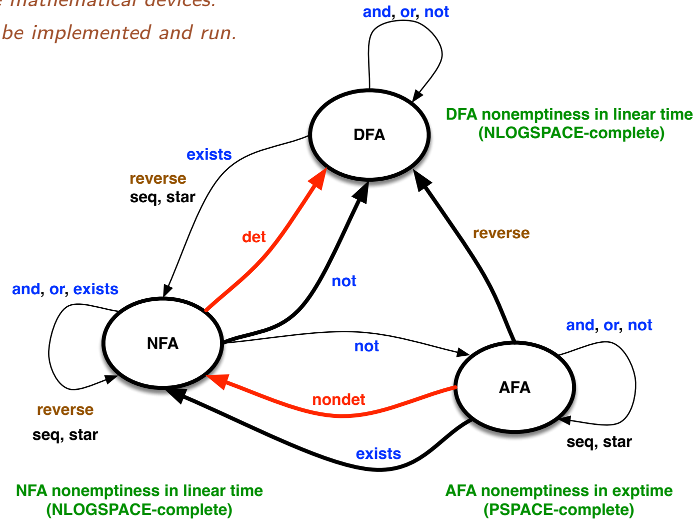
- in linear time if $\mathcal{A}_\varphi$ is an Alternating Finite-state Automata (AFA);
- in exponential time if $\mathcal{A}_\varphi$ is an Nondeterministic Finite-state Automaton (NFA);
- in double exponential time if $\mathcal{A}_\varphi$ is an Deterministic Finite-state Automaton (DFA).

We can compile reasoning into automata based procedures!

LTL$_f$ and Automata

Summary of automata theory on finite sequences:

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



LTL_f and Automata

Key point

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

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

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

We can compile reasoning into automata based procedures!

LTL_f and Automata

Alternating Automata on Finite Words (AFA)

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

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

AFA run

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

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

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

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

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

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

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

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

Negation Normal Form for LTL_f

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

NNF

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

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

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

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

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

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

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

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

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

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

$\textsc{ltl}_f$ fixpoint equations

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

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

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

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

Transition function δ

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

LTL_f fixpoint equations

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

LTL_f and Automata

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

NFA A_φ associated with an LTL_f formula φ (in NNF)

δ transition function

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

AFA2NFA transformation

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

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`.

LTL$_f$ and Automata

LTL$_f$ formulas can be directly translated in exponential time to NFAs, using AFA only implicitly.

NFA A_φ associated with an LTL$_f$ formula φ (in NNF)

Auxiliary rules

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

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

Algorithm: LTL$_f$2NFA

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

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`.

LTL$_f$ and Automata

In building the AFA and then 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:

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

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 $q' \models \delta(\Box A, A \wedge Last) = \delta(A, A \wedge Last) \wedge \delta(\bullet\Box A, A \wedge Last) = \texttt{true} \wedge \delta(\bullet\Box A, A \wedge Last)$, 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 $\neg Last$.)

LTL$_f$ and Automata: Examples

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

LTL_f and automata

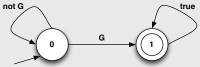
Key point

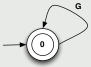
LTL_f formulas can be translated into a finite-state automaton on finite words $\mathcal{A}_\varphi$ such that:

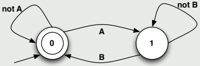
$$t \models \varphi \text{ iff } t \in \mathcal{L}(\mathcal{A}_\varphi)$$

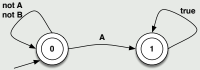
- in linear time if $\mathcal{A}_\varphi$ is an Alternating Finite-state Automata (AFA);
- in exponential time if $\mathcal{A}_\varphi$ is an Nondeterministic Finite-state Automaton (NFA);
- in double exponential time if $\mathcal{A}_\varphi$ is an Deterministic Finite-state Automaton (DFA).

Example (Automata for some LTL_f formulas)

$\Diamond G$


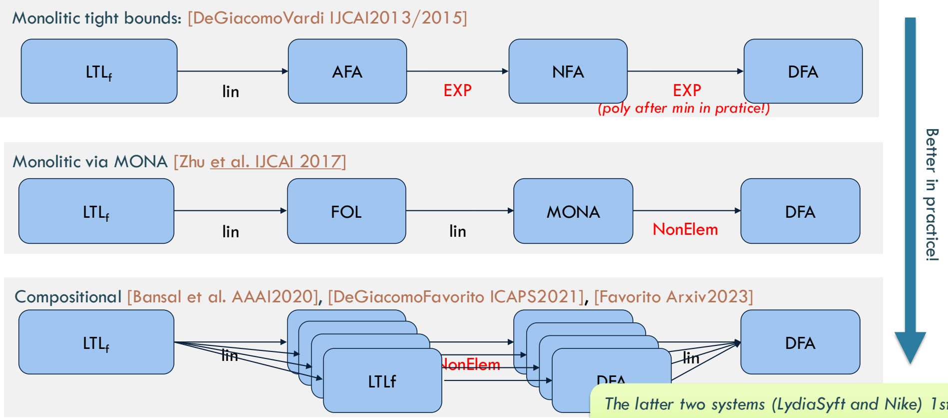
$\Box G$


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


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


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

LTL_f to Automata Techniques



The latter two systems (LydiaSyft and Nike) 1st and 2nd places in 1st edition LTLf synthesis track at SYNTHCOMP 2023 & 2024

LTL_f to Automata Techniques

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

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

On the fly forward fashion [Xiao et al. AAAI2021], [DeGiacomo et al. 2022], [Favorito Arxiv2023], [Xiao et al. Arxiv2024], [Duret-Lutz et al. 2025]

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

Based on "next normal form" or "progression" [BacchusKabanzaAAAI1998]:

eventually Red iff Red or next eventually Red

Important: transition must be "symbolic" i.e., propositional formulas

[Duret-Lutz et al. 2025] is implemented in SPOT, and is winner of the LTLf synthesis track at SYNTCOMP25

Outline

LTL_f Reasoning

LTL_f Satisfiability (φ SAT)

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

LTL_f Validity (φ VAL)

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

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

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

Thm: All the above reasoning tasks are PSPACE-complete. (Construction of NFA can be done while checking nonemptiness.)

As for the infinite traces.

Example

Example

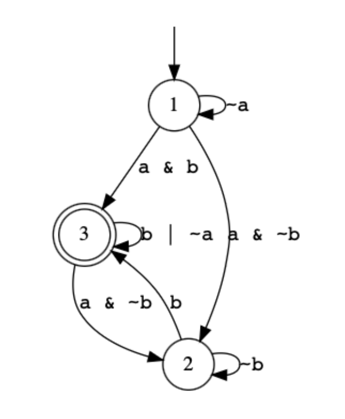
Question: Check whether $\varphi_1 = \Box(a \supset \Diamond b) \wedge \Diamond a$ is satisfiable. Answer:

- Compute the automaton for φ_1 by using e.g., `ltl2dfa` `<http://ltlf2dfa.diag.uniroma1.it/>` with input *G(a -> F b) & F a*.
- Check if there is a path from the initial state to a final state, e.g., by least fixpoint, using existential preimage: $PreE(\mathcal{E}) = \{q \mid \exists a, s'.\delta(s, a, s')$ and $s' \in \mathcal{E}\}$

$$\begin{aligned} Win_0 &= \{3\} \\ Win_1 &= Win_0 \cup \{1,2,3\} \\ Win_2 &= Win_1 \cup \{1,2,3\} \end{aligned}$$

Fixpoint!

- The initial state 1 is in Win, i.e, a final state is reachable from the initial one: the automaton accept at least a word and hence is nonempty.
- The formula φ_1 is satisfiable.



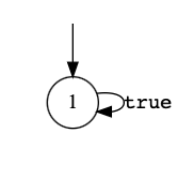
Example

Question: Check whether the logical implication $\Box(a \supset \Diamond b) \wedge \Diamond a \models \Diamond b$ holds. Answer:

- We solve it by satisfiability.
- $\Box(a \supset \Diamond b) \wedge \Diamond a) \models \Diamond b$ holds iff $\varphi_2 = \Box(a \supset \Diamond b) \wedge \Diamond a) \wedge \Box\neg b$ is unsatisfiable.
- Compute the automaton for φ_2 by using e.g., `ltl2dfa` `<http://ltlf2dfa.diag.uniroma1.it/>` with input *G(a -> F b) & F a & G !b* .
- Check if there is a path from the initial state to a final state, e.g., by least fixpoint, using existential preimage:
  $PreE(\mathcal{E}) = \{q \mid \exists a, s'.\delta(s, a, s') \text{ and } s' \in \mathcal{E}\}$

$$
\begin{aligned}
Win_0 &= \{\} \\
Win_1 &= Win_0 \cup \{\}
\end{aligned}
$$

Fixpoint!



- The initial state 1 is not in Win, i.e, no final states are reachable from the initial one: the automaton does not accept any word, i.e., is empty.
- The formula φ_2 is unsatisfiable, and hence the **logical implication holds**.

Example

Example

Consider the LTL_f formula Φ which is the conjunction of the following formulas:

$$\begin{array}{l} requested \wedge \\ \Box(requested \supset \Diamond acknowledged) \wedge \\ \Box(acknowledged \supset \bigcirc processed) \wedge \\ \Box(processed \supset \Diamond done) \end{array}$$

Questions:

- Check whether Φ is satisfiable.
- Check whether $\Phi \models \Diamond done$.

Use `ltl2dfa` <`http://ltlf2dfa.diag.uniroma1.it/`> to compute the automata, with input:

r & G(r -> F a) & G(a ->X p) & G(p -> F d)); r & G(r -> F a) & G(a ->X p) & G(p -> F d) & ! F d)

Outline

From Transition Systems to Automata

TS to Automata

A (state-labelled) transition system $\mathcal{T} = (S, s_0, \rightarrow, \lambda)$ with $\rightarrow \subseteq S \times S$ and $\lambda : S \rightarrow 2^{\mathcal{P}}$ is turned into a NFA $\mathcal{A}_{\mathcal{T}} = (\Sigma, Q, Q_0, \delta, F)$ where:

- $\Sigma = 2^{\mathcal{P}}$
- $Q = S \cup \{init\}$
- $Q_0 = \{init\}$
- $\delta \subseteq Q \times \Sigma \times Q$
  - $(init, \sigma, s_0) \in \delta$ if $\sigma = \lambda(s_0)$
  - $(s, \sigma, s') \in \delta$ if $s \rightarrow s'$ and $\sigma = \lambda(s')$
- $F = S$ (or even $F = Q$).

If the labeling of the resulting states of transitions is always different, this construction returns a DFA.

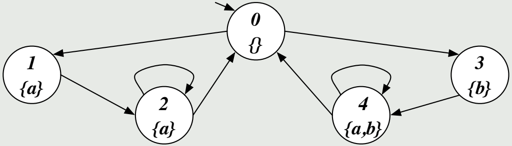
Intuitively:

- The labeling of states is pushed backward to the incoming edges.
- A root state is included to push the initial state labels backward.
- Every state, possibly except ϵ, is accepting.

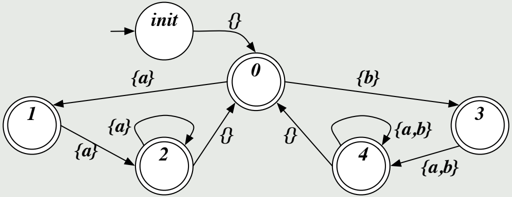
Example

Example

Transition system:



Corresponding DFA:



LTL$_f$ Model Checking

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

LTL$_f$ model checking algorithm

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

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

The same results holds for LTL on infinite traces.

Reminder: Cartesian Product/Intersection of Automata

Cartesian Product/Intersection of Automata

Let $A_1 = (\Sigma, S_1, s_1^0, \rho_1, F_1)$ and $A_2 = (\Sigma, S_2, s_2^0, \rho_2, F_2)$. Intuitively, we construct the **intersection (or AND)** $A_1 \times A_2$, also called (synchronous) **Cartesian product**, that runs simultaneously both A_1 and A_2 on the input word and accepts when **both** accept. We define $A_1 \times A_2$ as:

$$A_1 \times A_2 = (\Sigma,\ S_1 \times S_2,\ (s_1^0, s_2^0),\ \rho,\ F_1 \times F_2)$$

where:

- The **transition function** requires to execute the two automata **concurrently** in a **synchronized** way:

  $$((s_1, s_2), a, (s_1', s_2')) \in \rho \quad \text{iff} \quad (s_1, a, s_1') \in \rho_1 \text{ and } (s_2, a, s_2') \in \rho_2$$

- The condition defining the **final states**:

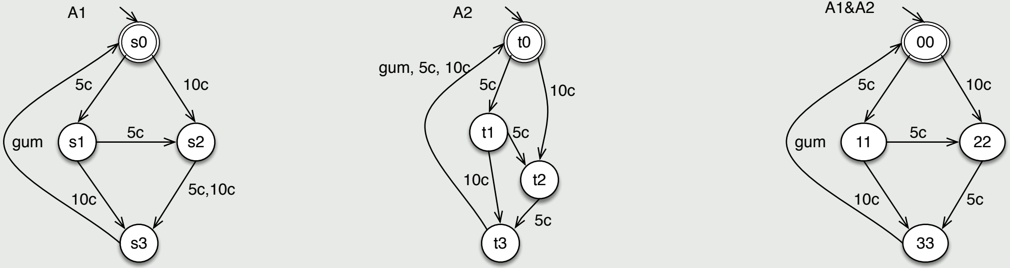
  $$F_1 \times F_2$$

  says that we accept a word if **both** automata A_1 and A_2 accept it.

It is easy to see that the size of $A_1 \times A_2$ is the product of the sizes of A_1 and of A_2, and that $\mathcal{L}(A_1 \times A_2) = \mathcal{L}(A_1) \cap \mathcal{L}(A_2)$.

Example

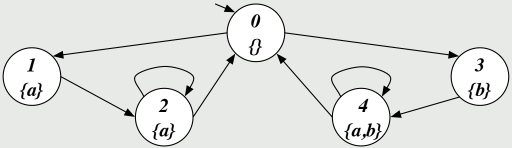
Example (AND/intersection of DFA's)



Example

Example

Consider the transition system in figure



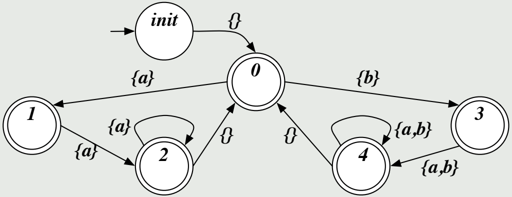
and the property $\varphi = \Box((\neg a \land \neg b) \supset \bullet\bullet a)$

Does $\mathcal{T} \models \varphi$?

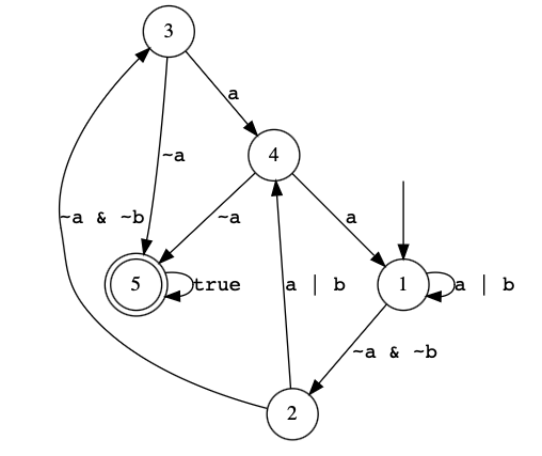
Example

Example (Solution)

DFA $\mathcal{A}_{\mathcal{T}}$:



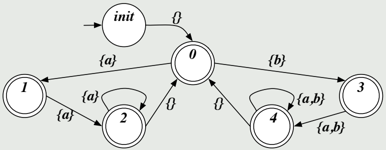
$\mathcal{A}_{\neg\varphi}$ (where $\neg\varphi = \neg\Box((\neg a \wedge \neg b) \supset \bullet\bullet a)$)



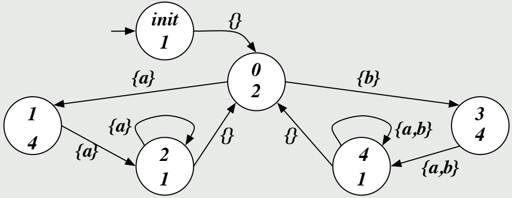
Example

Example (Solution)

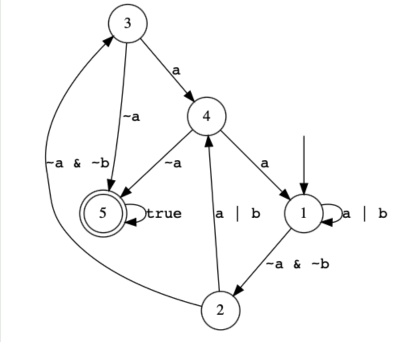
DFA $\mathcal{A}_{\mathcal{T}}$:



Cartesian product $\mathcal{A}_{\mathcal{T}} \times \mathcal{A}_{\neg\varphi}$. i.e. $\mathcal{A}_{\mathcal{T}} \wedge \mathcal{A}_{\neg\varphi}$ (only reachable part shown):



$\mathcal{A}_{\neg\varphi}$

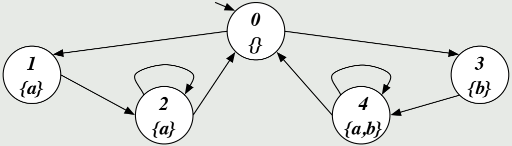


Note: no final state (in the reachable part) so no finite trace belongs both to $\mathcal{T}$ and to $\neg\varphi$.
Hence all finite traces of $\mathcal{T}$ satisfy φ, i.e., $\mathcal{T} \models \varphi$ holds.

Example

Example

Consider the transition system in figure



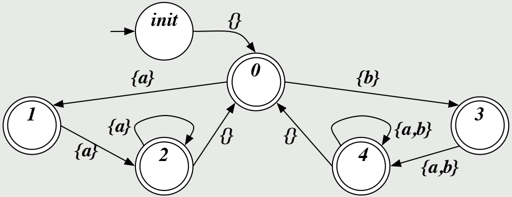
and the property $\varphi = \Box((\neg a \wedge \neg b) \supset \bigcirc\bigcirc a)$

Does $\mathcal{T} \models \varphi$?

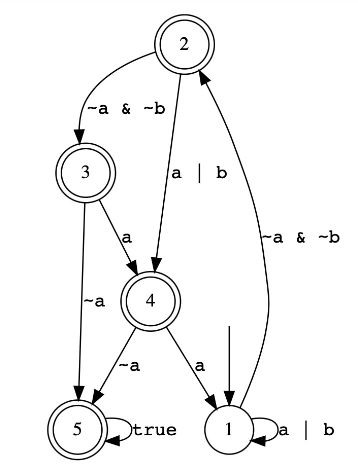
Example

Example (Solution)

DFA $\mathcal{A}_{\mathcal{T}}$:



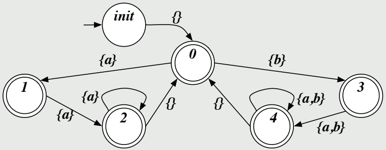
$\mathcal{A}_{\neg\varphi}$ (where $\neg\varphi = \neg\Box((\neg a \wedge \neg b) \supset \bigcirc\bigcirc a)$)

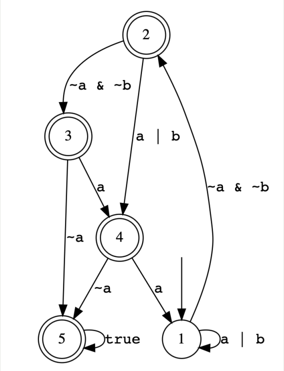


Example

Example (Solution)

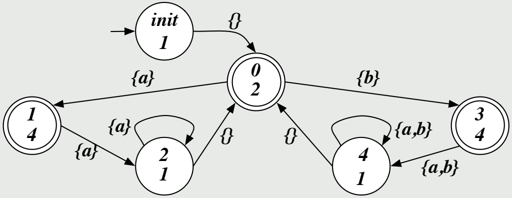
DFA $\mathcal{A}_\mathcal{T}$:





$\mathcal{A}_{\neg\varphi}$

Cartesian product $\mathcal{A}_\mathcal{T} \times \mathcal{A}_{\neg\varphi}$. i.e. $\mathcal{A}_\mathcal{T} \wedge \mathcal{A}_{\neg\varphi}$ (only reachable part shown):



Note we have final states (in the reachable part) so there exist finite traces that belongs both to $\mathcal{T}$ and to $\neg\varphi$. Hence it is not that case that all finite traces of $\mathcal{T}$ satisfy φ, i.e., $\mathcal{T} \models \varphi$ does not holds.

Outline

Model Checking Planning Domains

What about model checking planning domains? After all they are also transition systems.

Planning domains induce transition systems that are labelled both on the transitions and the states.

Transition System Induced by a Domain

A domain $D = (\mathcal{F}, \mathcal{A}, I)$ induces the transition system $T_D = (\mathcal{S}, \mathcal{A}, s_0, \alpha, \delta)$ where:

- $\mathcal{F}$ is the **fluents** (atomic propositions)
- $\mathcal{A}$ is the **actions** (atomic symbols)
- $\mathcal{S} = 2^{\mathcal{F}}$ is the set of states
- s_0 is the initial state (initial assignment to fluents)
- $\alpha(s) \subseteq \mathcal{A}$ represents **action preconditions**
- $\delta(s, a, s')'$ with $a \in \alpha(s)$ represents **action effects (including frame)** –if deterministic the δ is a function instead of a relation.

Model Checking Planning Domains

We can rewrite the transitions system induced by the domain into a standard **labelled transition system**, labelled on both transitions and states.

Labelled Transition System Induced by a Domain

A domain $D = (\mathcal{F}, \mathcal{A}, I)$ induces the transition system labeled on both transition and states $T_D = (\mathcal{S}, s_0, \dot{\longrightarrow}, \lambda)$ where:

- $\mathcal{S} = 2^{\mathcal{F}}$ is the set of states
- s_0 is the initial state
- $\dot{\longrightarrow} \subseteq S \times \mathcal{A} \times \mathcal{S}$ such that $s \xrightarrow{a} s'$ with $a \in \alpha(s)$ and $\delta(s, a, s')$
- $\lambda : S \to 2^{\mathcal{F}}$ such that $\lambda(s) = s$.

TS-Traces

A **trace** for T_D is a finite sequence:

$$s_0, a_1, s_1, \cdots, a_n, s_n$$

where there exists a sequence of states $s_0, \ldots, s_n$ where s_0 is the initial state $s_i, \xrightarrow{a_{i+1}} s_{i+1}$ for $i = 0, \ldots, n-1$; and $s_i = \lambda(s_i)$ for $i = 0, \ldots, n$.

Model Checking Planning Domains

(1) In order to talk about such transition system in LTL_f –or LTL for the matter– we need to:

Represent actions as propositions

Decide how we represent actions as propositions of LTL_f formulas.

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

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

Model Checking Planning Domains

(2) In order to talk about such transition system in LTL_f –or LTL for the matter– we also need to:

Pair actions and states in a time instant

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

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

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

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

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

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

Model Checking Planning Domains

LTL_f-Traces

A T_D trace $s_0, a_1, s_1, \cdots, a_n, s_n$ induces a corresponding LTL_f-trace:

- If we pair action and the resulting state: $(dummy, s_0), (a_1, s_1), \cdots, (a_n, s_n)$, where $dummy$ is a dummy starting action.
- If we pair state and the next action: $(s_0, a_1), (s_1, a_2) \cdots, (s_{n-1} a_n), (s_n, dummy)$, where $dummy$ is a dummy ending action.

Example

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

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

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

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

From Labelled Transition Systems to Automata

TS to Automata

A (transition and state) labelled transition system $\mathcal{T} = (S, s_0, \stackrel{\cdot}{\longrightarrow}, \lambda)$ with $\stackrel{\cdot}{\longrightarrow} \subseteq S \times \mathcal{A} \times \mathcal{S}$ and $\lambda : S \rightarrow 2^{\mathcal{P}}$ is turned into a NFA $\mathcal{A}_{\mathcal{T}} = (\Sigma, Q, Q_0, \delta, F)$ where:

- $\Sigma = 2^{\mathcal{P}}$
- $Q = S \cup \{init\}$
- $Q_0 = \{init\}$
- $\delta \subseteq Q \times \Sigma \times Q$
  - $(init, \sigma, s_0) \in \delta$ if $\sigma = \lambda(s_0) \cup \{dummy\}$
  - $(s, \sigma, s') \in \delta$ if $s \xrightarrow{a} s'$ and $\sigma = \lambda(s') \cup \{a\}$
- $F = S$ (or even $F = Q$).

Observe: *In the case of planning domains the labeling of the resulting states of transitions is always different, hence this construction returns a* DFA.

Intuitively:

- The labeling of states is pushed backward to the incoming edges together with the labeling of the transitions.
- A root state is included to push the initial state labels backward, together with a $dummy$ action.
- Every state, possibly except ϵ, is accepting.

LTL_f Model Checking

The same model checking algorithm introduced before now can be applied to handle (transition and state) labelled transition systems, and hence domains.

LTL_f model checking algorithm

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

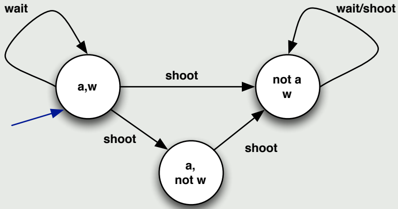
Thm: Verification is PSPACE-complete, and most importantly polynomial in the transition system.

The same results holds for LTL on infinite traces.

Example

Example

Consider the planning domain in figure



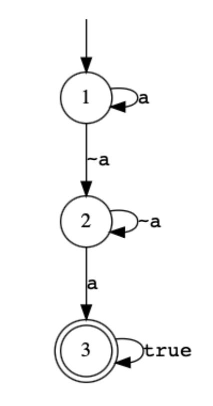
and the property $\varphi = \Box(\neg a \supset \Box\neg a)$

Does $\mathcal{T} \models \varphi$?

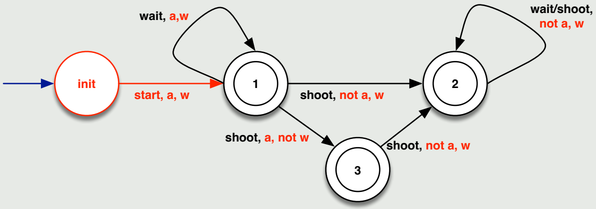
Example

Example (Solution)

$\mathcal{A}_{\neg\varphi}$ (where $\Box(\neg a \supset \Box\neg a)$)

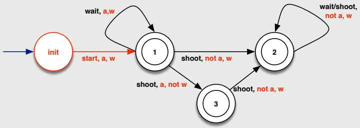


DFA $\mathcal{A}_{\mathcal{T}}$:



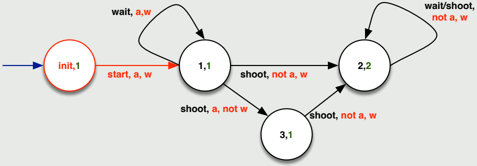
Example

Example (Solution)

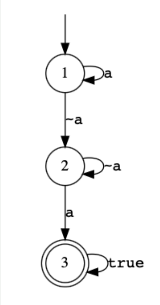


DFA $\mathcal{A}_{\mathcal{T}}$:

Cartesian product $\mathcal{A}_{\mathcal{T}} \times \mathcal{A}_{\neg\varphi}$. i.e. $\mathcal{A}_{\mathcal{T}} \wedge \mathcal{A}_{\neg\varphi}$ (only reachable part shown):



$\mathcal{A}_{\neg\varphi}$

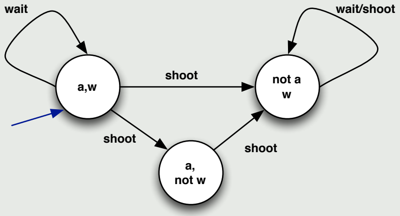


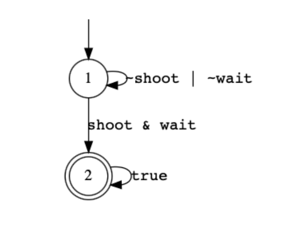
Note: no final state (in the reachable part) so no finite trace belongs both to $\mathcal{T}$ and to $\neg\varphi$.
Hence all finite traces of $\mathcal{T}$ satisfy φ, i.e., $\mathcal{T} \models \varphi$ holds.

Example

Example

Consider the planning domain in figure





and the property that only one action is executed at each time, i.e., $\varphi = \Box(shoot \supset \neg wait)$

Does $\mathcal{T} \models \varphi$?

Observe: this is going to be true by construction since we have one action at the time in the domain and we never add other actions in the transitions of the automaton. But it is easy to check it by model checking considering that the automaton for $\neg\varphi$ is the one in figure of the right.