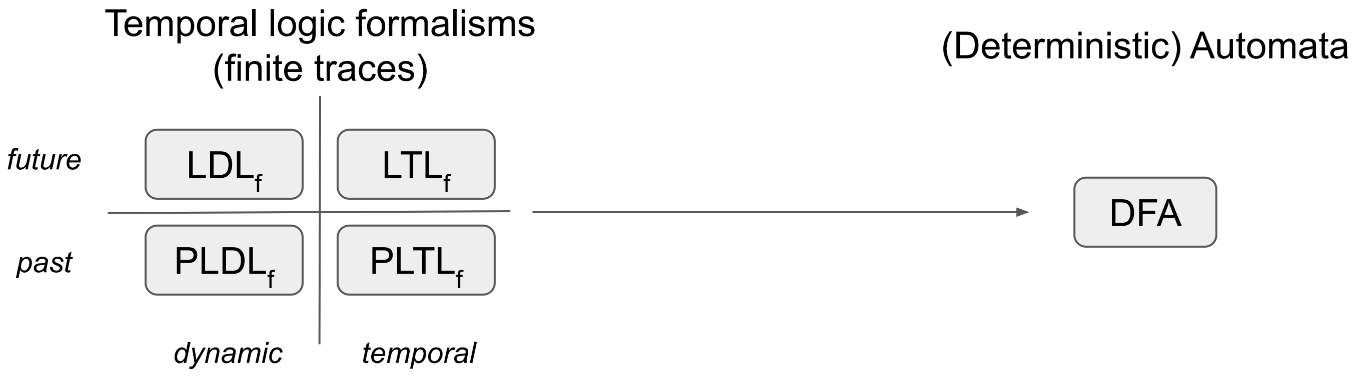
LDL_f/LTL_f-to-DFA in practice

Marco Favorito
(PhD student)

Recommended prerequisite:

- Slides on “DFA, NFA, AFA on finite words”
- Slides on “Linear Temporal Logics on Finite Traces: LTLf and LDLf”

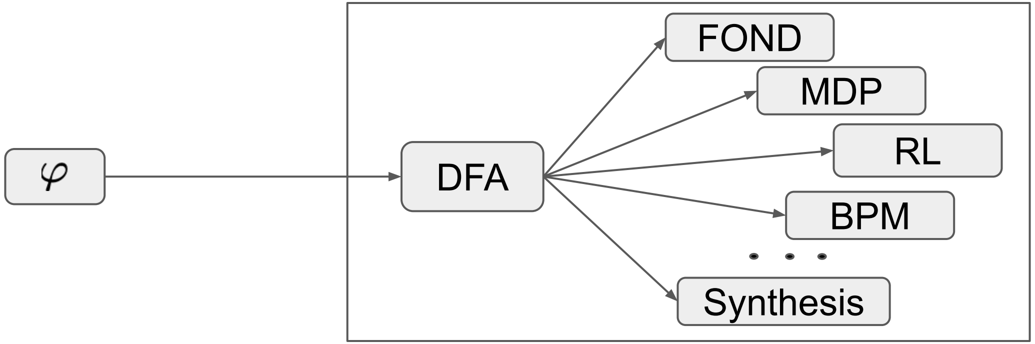
The Problem



Temporal logic formalisms
(finite traces)
future
past
LDL_f
LTL_f
$PLDL_f$
$PLTL_f$
dynamic
temporal
(Deterministic) Automata
DFA

Why care?

- Many applications in AI and CS, based on DFAs



- Often easier to work with logic than with automata
  - Logics are closer to natural language

Semantics: finite traces of propositional interpretations

Given a set of propositions $\mathcal{P}$, traces are sequences of *propositional interpretations* $(2^{\mathcal{P}})^*$

E.g. from the Yale Shooting domain:

$$\mathcal{P} = \{alive, working\}$$

$$2^{\mathcal{P}} = \{\emptyset, \{alive\}, \{working\}, \{alive, working\}\}$$

An example of trace:

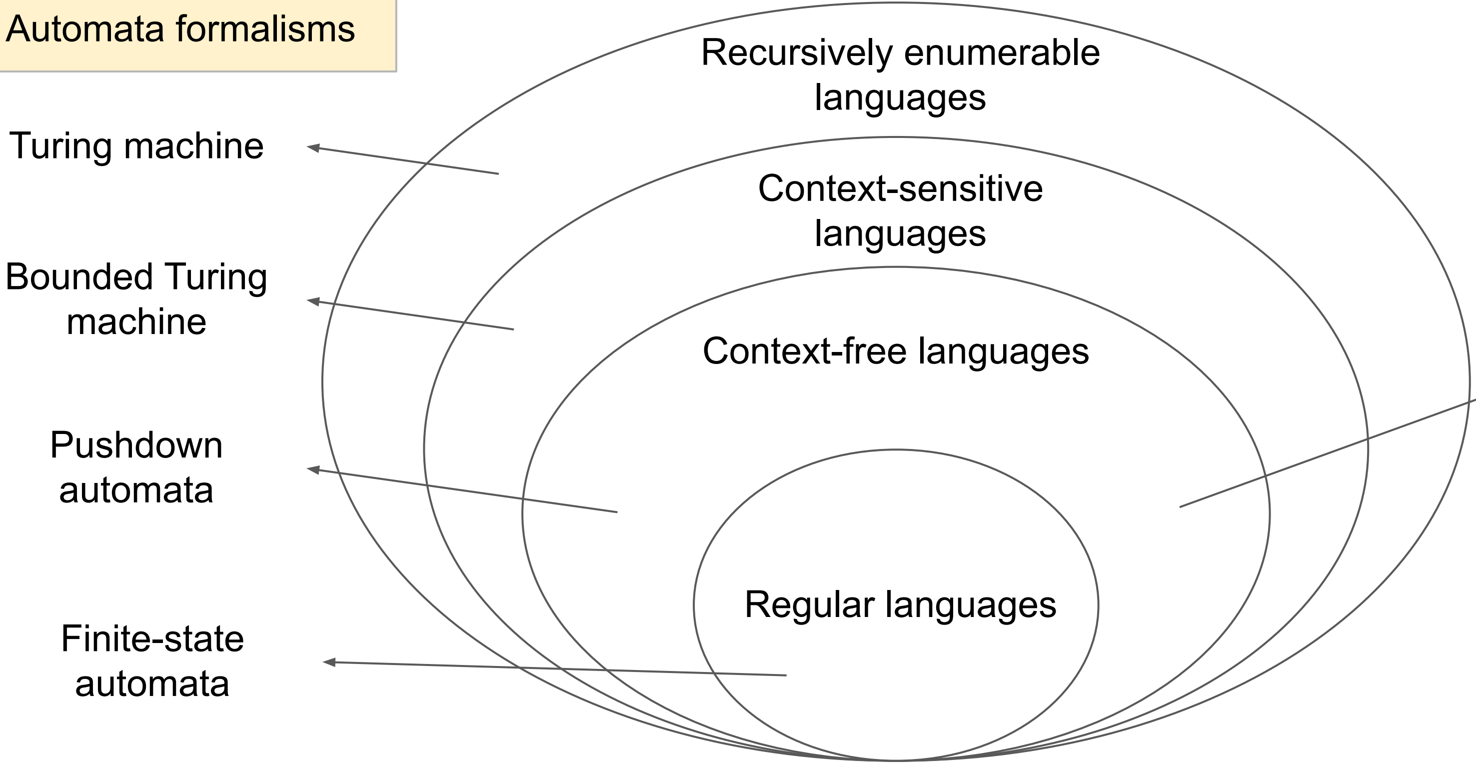
$$\pi = \{alive, working\}, \{alive, working\}, \{alive\}, \{alive\}, \{working\}$$

Set of traces <-> Language

AI view	Language-theoretic view
set of prop. int. $2^{\mathcal{P}}$	alphabet Σ
trace $\pi \in (2^{\mathcal{P}})^*$	word $w \in \Sigma^*$
set of traces $\Pi \subseteq (2^{\mathcal{P}})^*$	Language (set of words) $\mathcal{L} \subseteq \Sigma^*$

Chomsky hierarchy

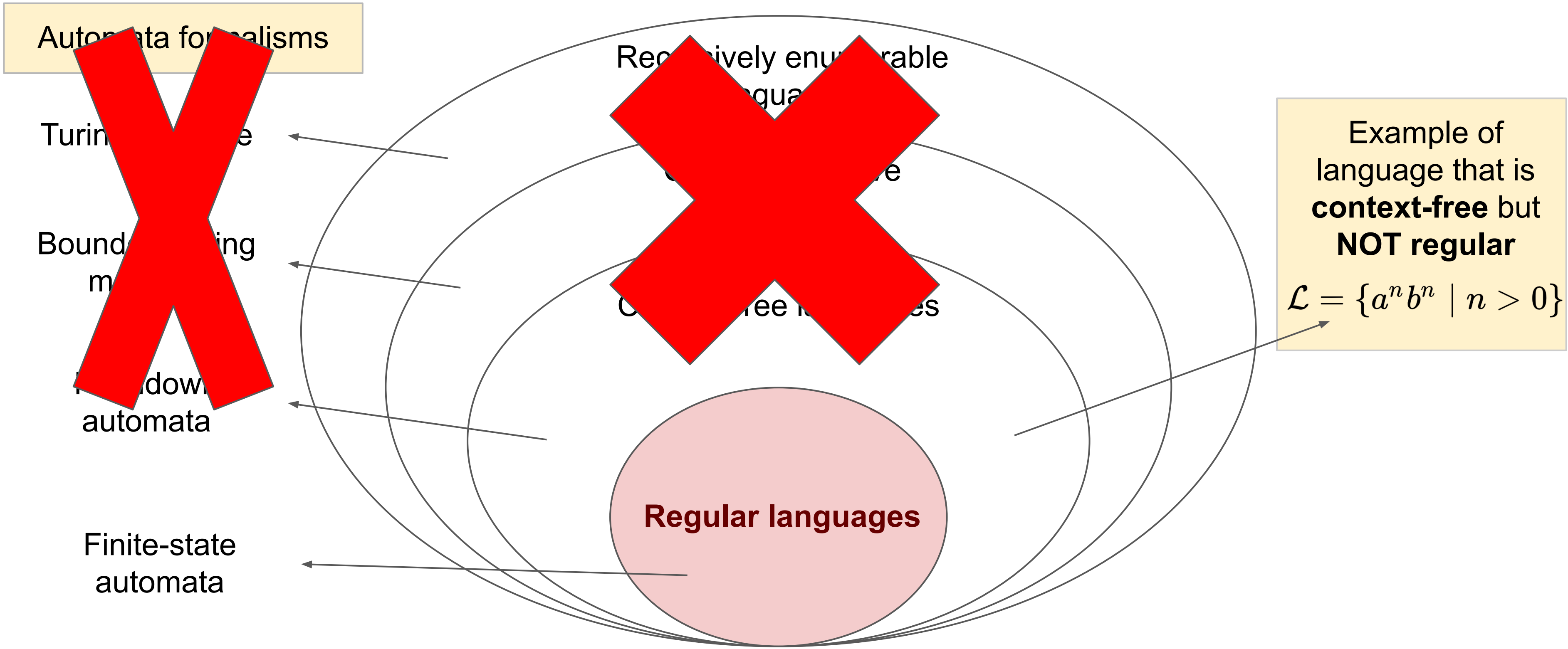
Automata formalisms



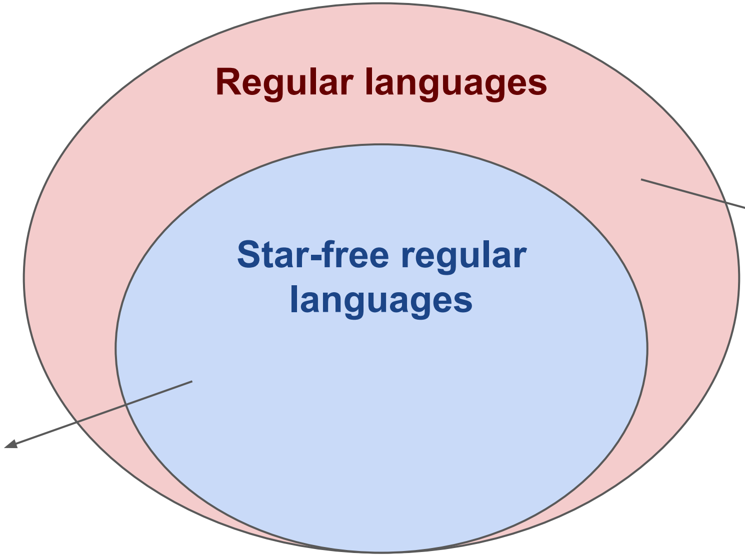
Example of language that is **context-free** but **NOT regular**

$$\mathcal{L} = \{a^n b^n \mid n > 0\}$$

Chomsky hierarchy



Logics, automata and (regular) languages



Example of language that is **regular** but **NOT star-free regular**

$$\mathcal{L} = \{(aa)^n \mid n \geq 0\}$$

(words of even length)

Example of **star-free regular** language

$$\mathcal{L} = \{w \mid w \models true^* a \, true^*\}$$

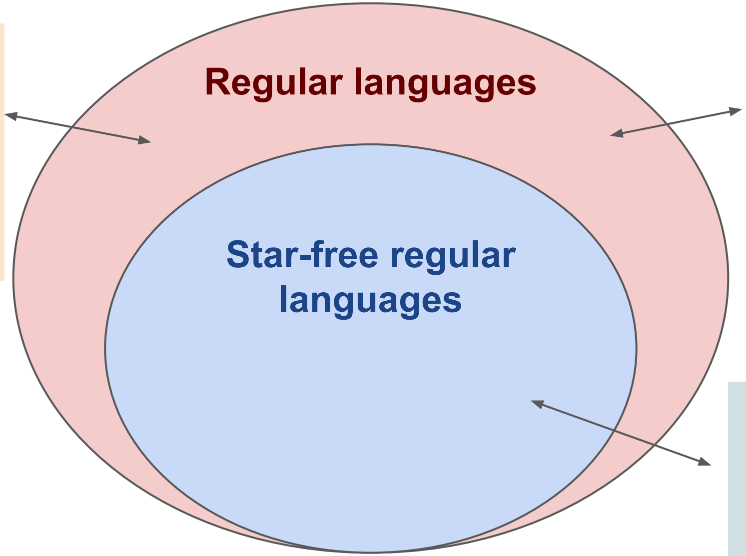
(liveness)

Logics, automata and (regular) languages

Deterministic Finite Automata (DFA)

Nondeterministic Finite Automata (NFA)

Alternating Finite Automata (AFA)



Monadic Second-order logic (MSO)

Regular Expressions (RE)

Linear Dynamic Logic (LDL)

Past Linear Dynamic Logic (PLDL)

First-order Logic (FOL)

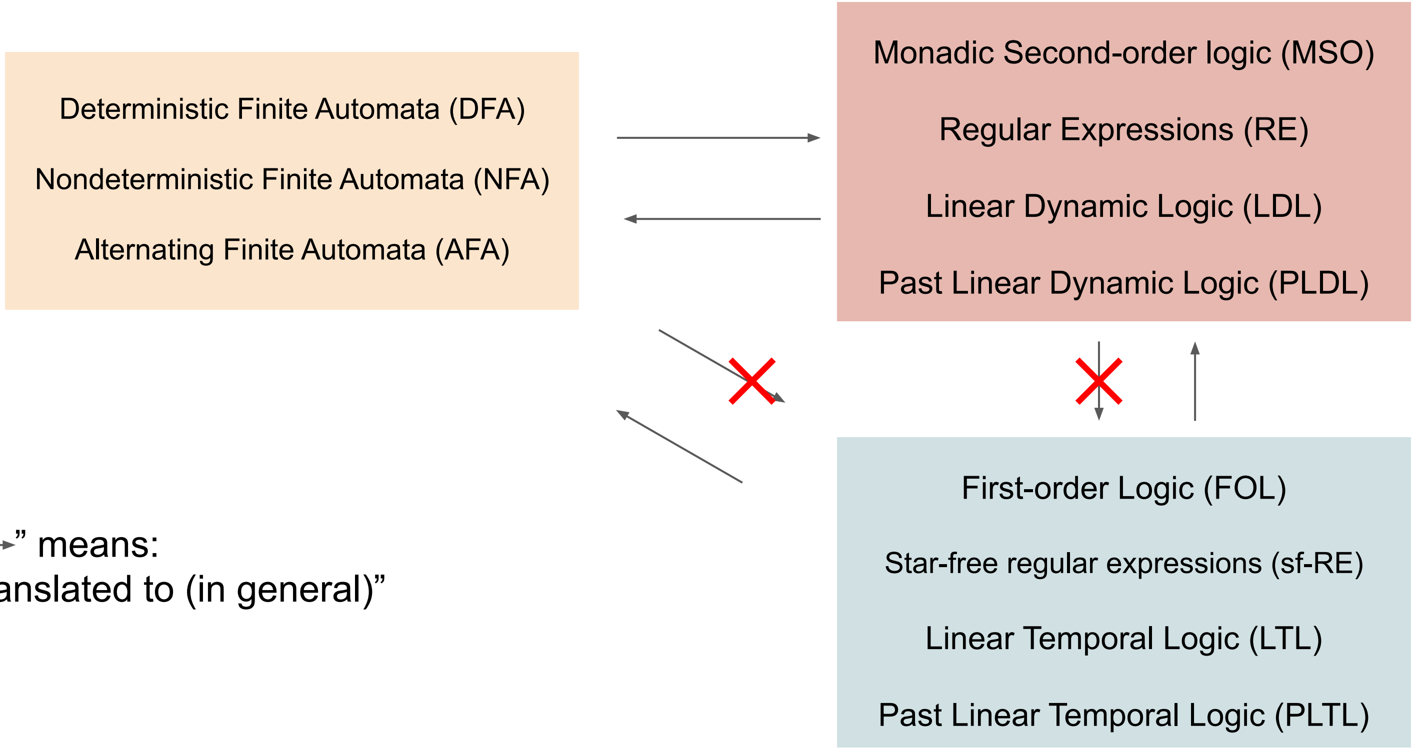
Star-free regular expressions (sf-RE)

Linear Temporal Logic (LTL)

Past Linear Temporal Logic (PLTL)

Note:
Automata with finite, constant memory

Logics, automata and (regular) languages



The "⟶" means:
"can be translated to (in general)"

From logics to automata: computational complexity

$$\text{MSO} \xrightarrow[\text{(nonelementary)}]{} \text{DFA}$$

Nonelementary is bad

- size of DFA cannot be upper-bounded a priori wrt any formula
- Arbitrary tower of exponentials: $2^{2^{\cdot^{\cdot^{\cdot^{2^n}}}}}$
  - Still: <u>it works well in practice!</u> (see later)

From logics to automata: computational complexity

$$\text{MSO} \xrightarrow[\text{(nonelementary)}]{} \text{DFA}$$

$$\text{RE} \xrightarrow[\text{(exponential)}]{} \text{DFA}$$

- (only one) exponential blow-up - **good**
- Regexes are **NOT** closed under <u>negation</u> and <u>conjunction</u> - **bad**
- <u>Negation</u> requires an exponential blow-up - **bad**

From logics to automata: computational complexity

$$\text{MSO} \longrightarrow \text{DFA}$$

(nonelementary)

$$\text{RE} \longrightarrow \text{DFA}$$

(exponential, but...)

(De Giacomo and Vardi, 2013):

$$\text{LDL}_f \xrightarrow[\text{(linear)}]{} \text{AFA} \xrightarrow[\text{(exponential)}]{} \text{NFA} \xrightarrow[\text{(exponential)}]{} \text{DFA}$$

- $\text{LDL}_f \approx \text{LTL}_f$ + Regular Expressions - **good**
- Closed under negation - **good**
- Complexity is double-exponential: 2^{2^n} - **fair enough**
  - The same holds for LTL_f

From logics to automata: computational complexity

$$\text{MSO} \xrightarrow{\text{(nonelementary)}} \text{DFA}$$

$$\text{RE} \xrightarrow{\text{(exponential, but...)}} \text{DFA}$$

$$\text{LDL}_f \longrightarrow \text{AFA} \longrightarrow \text{NFA} \longrightarrow \text{DFA}$$

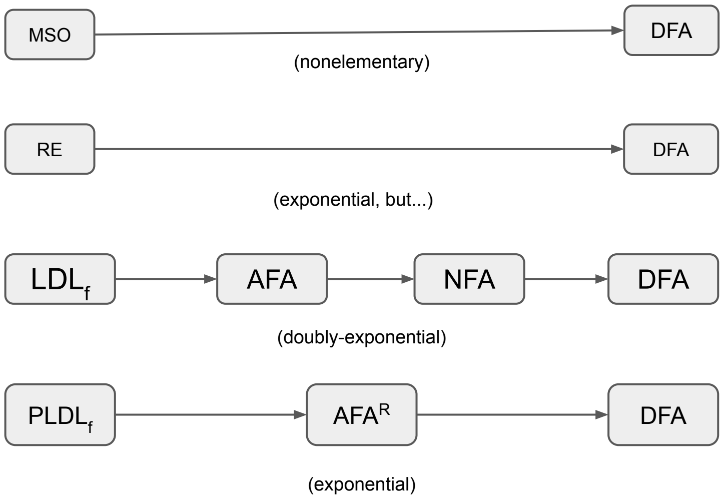
(doubly-exponential)

(De Giacomo, Di Stasio, Fuggitti, Rubin, 2020):

$$\text{PLDL}_f \xrightarrow{\text{(linear)}} \text{AFA}^R \xrightarrow{\text{(exponential)}} \text{DFA}$$

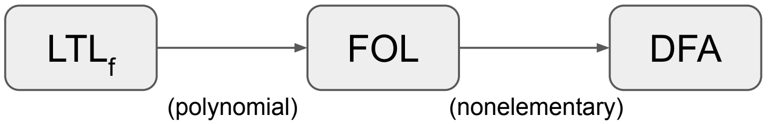
- One exponential - **good**
- PLDL_f to LDL_f costs 2EXP - **bad**
    - we can only work within PLDL_f

From logics to automata: computational complexity



Other approaches

- Encoding of LTL_f into FOL (Zhu et al., 2017), (Bansal et al. 2020):



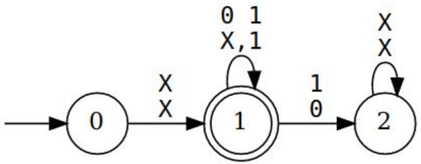
- From LDL_f directly to DFA (De Giacomo and Favorito, 2021)

MSO -> DFA: The MONA tool

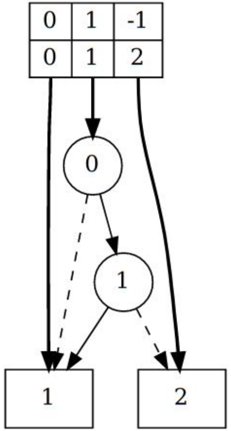
<u>MONA</u> is a C library and tool for translating MSO (and hence FOL too) formulae to DFA.

- [cs-au-dk/MONA](https://github.com/cs-au-dk/MONA)
- DFAs in MONA are represented by shared, multi-terminal BDDs.
    - The representation is ***explicit*** in the state space,
    - and ***symbolic*** in the transitions

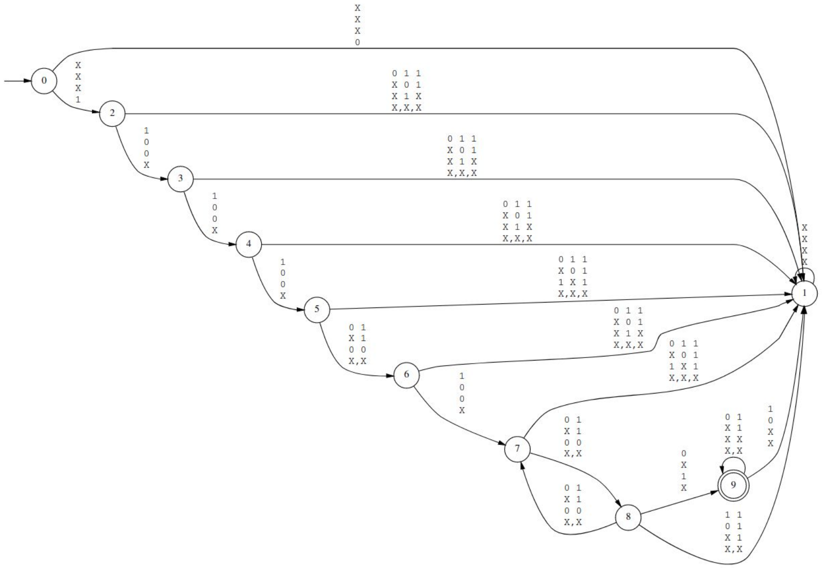
Classic representation:

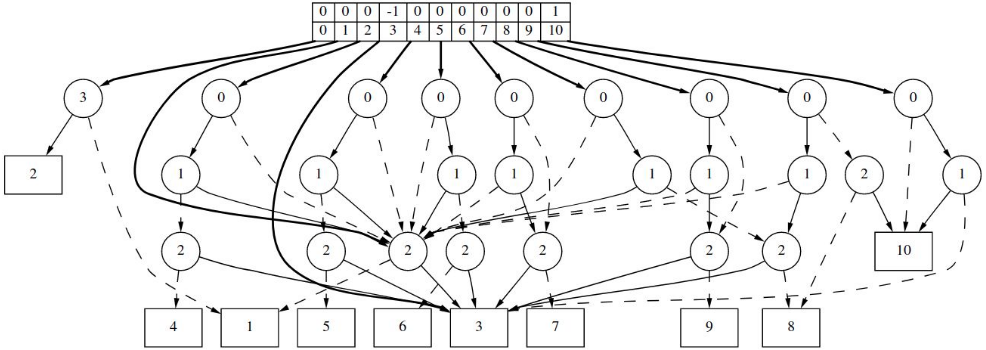


MONA DFA data structure:



MONA DFA



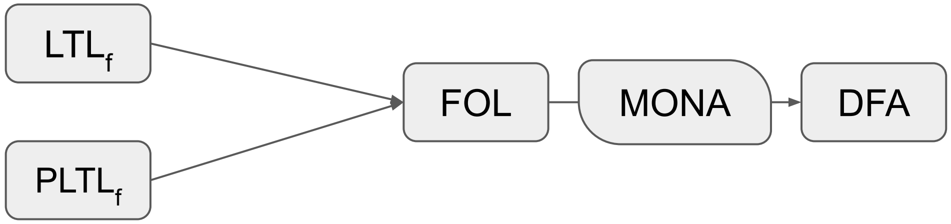


Classic representation
16 (2^4) letters and 10 states
Transition table entries: 160

MONA DFA
multi-terminal, shared Binary Decision Diagram
Acyclic, directed graph with only 35 nodes

LTL_f/$PLTL_f$ -> FOL -> DFA (Zhu et al. 2017)

Encoding of LTL_f into FOL, and then use MONA



Implementations:

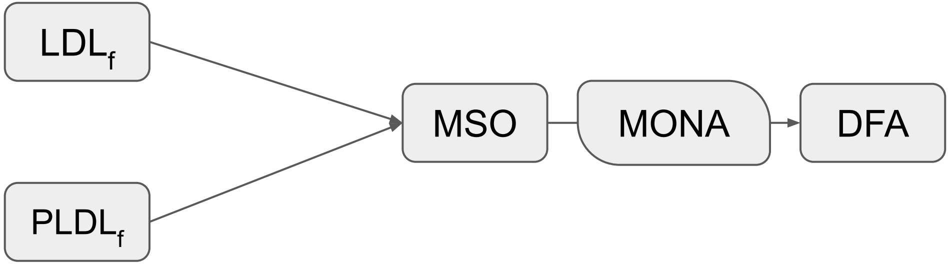
- Syft "ltlf2fol" + MONA: https://github.com/Shufang-Zhu/Syft
  - Written in C++
  - Used by Lisa (Bansal et al. 2020)
- LTLf2DFA (also supports PLTLf):
  - Written in Python, uses MONA
  - GitHub: https://github.com/whitemech/LTLf2DFA/
  - Web app: http://ltlf2dfa.diag.uniroma1.it/

Encoding of LTLf into MSO?

Shown to perform worse than FOL encoding:

First-Order vs. Second-Order Encodings for LTLf-to-Automata Translation (Zhu et al. 2019)

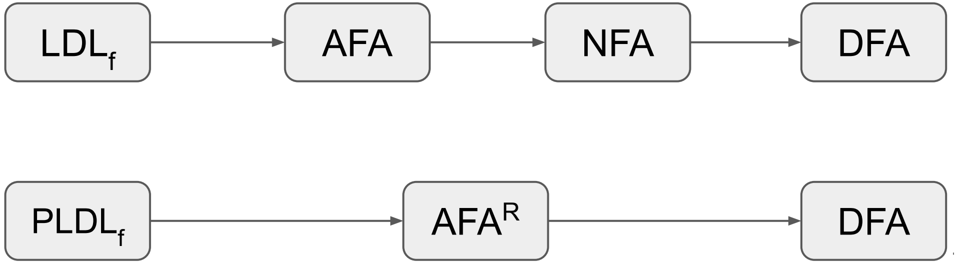
$LDL_f/PLDL_f$ -> MSO -> DFA



- Encodings from $LDL_f/PLDL_f$ to MSO
- Then, use MONA to compute the DFA

<u>Not done yet!</u>
(Possible topic for projects/theses)

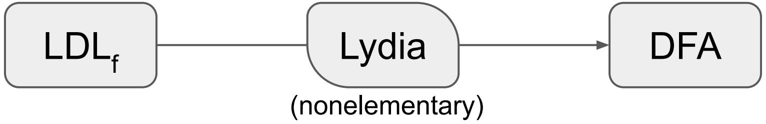
LDL_f/$PLDL_f$ -> AFA -> NFA -> DFA



<u>No scalable implementations exist!</u>
(Possible topic for projects/theses)

LDL_f -> DFA (De Giacomo and Favorito, 2021)

- From LDL_f directly to DFA



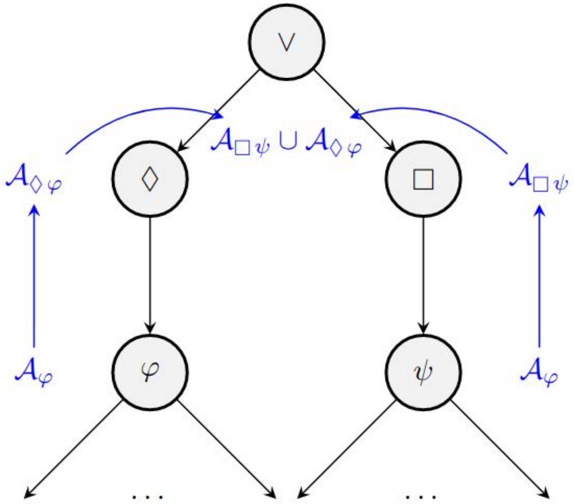
- Implemented in the **Lydia** tool:
  - Uses the MONA DFA representation (but not MSO)
  - GitHub repo: https://github.com/whitemech/lydia
  - Web app: https://lydia.whitemech.it

- Compositional: breaks down formulae in smaller parts and compute their DFA
- NONELEMENTARY (instead of best theoretical bound of 2EXPTIME)
  - But works fairly well in practice

How Lydia works (TL;DR)

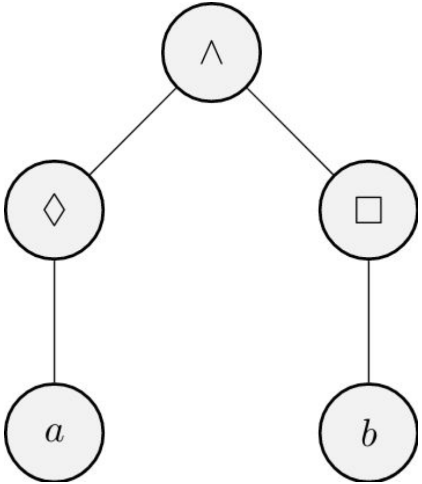
- Mapping from LDLf operators to DFA operations
- Inductively apply these mappings
- If we encounter LTLf formulae, translate them in LDLf

E.g. $\Diamond \varphi \vee \Box \psi$



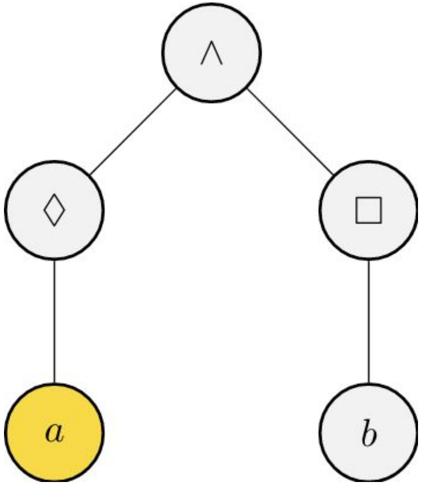
Example

$$\Diamond a \land \Box b$$

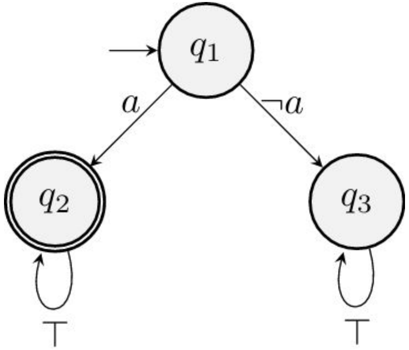


How to compute $\mathcal{A}_{\Diamond a \land \Box b}$?

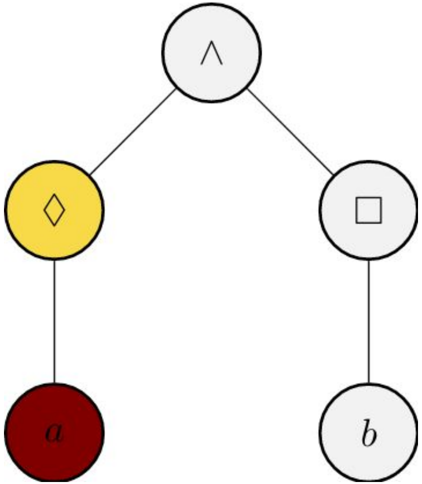
Example



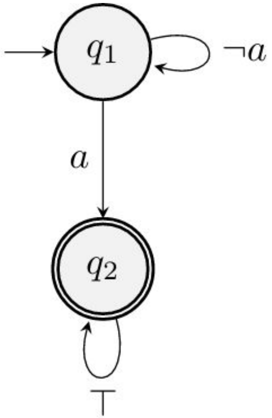
$\mathcal{A}_a$



Example

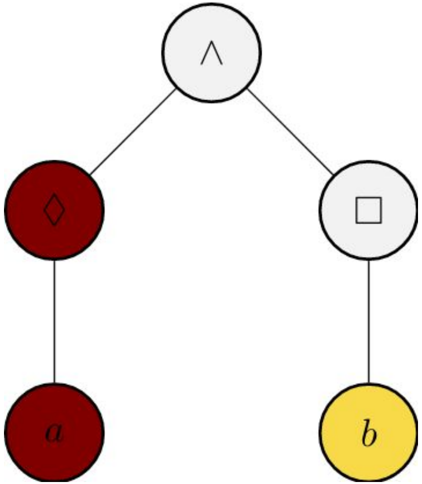


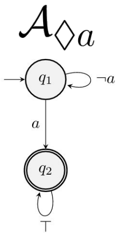
$$f(\Diamond, \mathcal{A}_a) = \mathcal{A}_{\Diamond a}{}^{*}$$

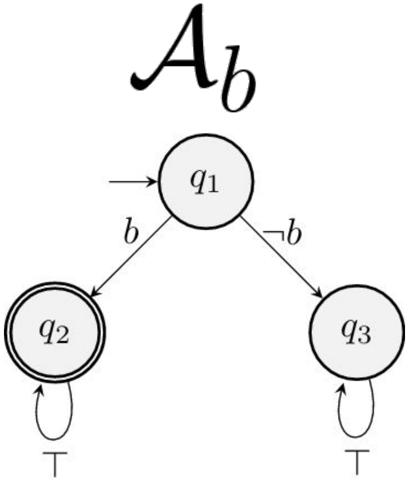


* $\Diamond\varphi \equiv \langle true^* \rangle(\varphi \wedge \neg end)$

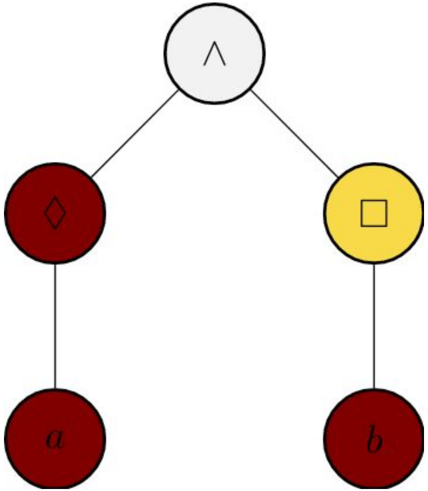
Example


$\wedge$
$\Diamond$
$\Box$
a
b

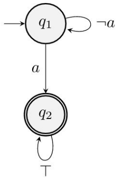

$\mathcal{A}_{\Diamond a}$
q_1
$\neg a$
a
q_2
$\top$


$\mathcal{A}_b$
q_1
b
$\neg b$
q_2
q_3
$\top$
$\top$

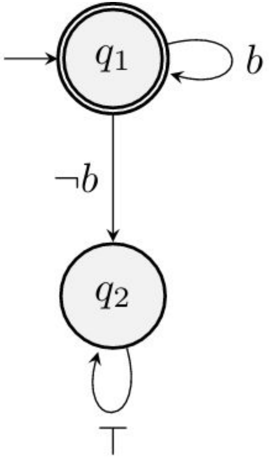
Example



$\mathcal{A}_{\Diamond a}$



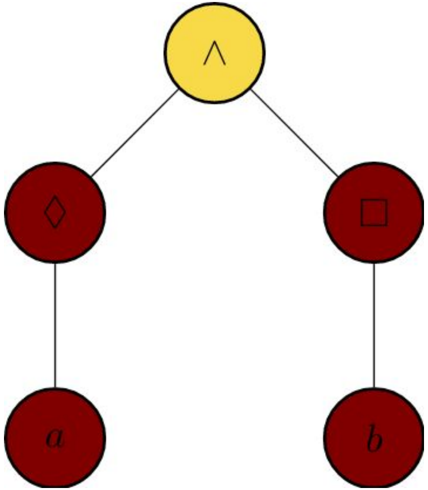
$$f(\Box, \mathcal{A}_b) = \mathcal{A}_{\Box b}\ ^*$$

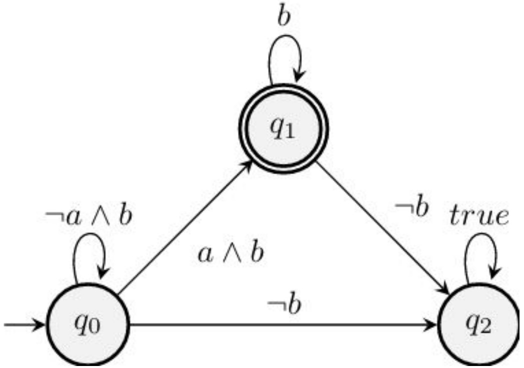


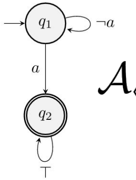
$^*\ \Box\varphi \equiv [true^*](\varphi \vee end)$

Example

$$f(\wedge, \mathcal{A}_{\Diamond a}, \mathcal{A}_{\Box b}) = \mathcal{A}_{\Diamond a} \cap \mathcal{A}_{\Box b}$$

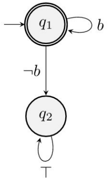






$\mathcal{A}_{\Diamond a}$

$\mathcal{A}_{\Box b}$



LDLf syntax

- We use the LDLf syntax that works for empty traces (Brafman, De Giacomo, and Patrizi, 2018)
- Given a set of propositional symbols *P*, LDLf formulae are built as follows:

$$
\begin{aligned}
\varphi &::= \mathit{tt} \mid \mathit{ff} \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \langle\rho\rangle\varphi \mid [\rho]\varphi \\
\rho &::= \phi \mid \varphi? \mid \rho_1 + \rho_2 \mid \rho_1 ; \rho_2 \mid \rho^*
\end{aligned}
$$

- Where φ is a propositional formula over *P*

LTLf -> LDLf (linear)

$$tr(\phi) = \langle\phi\rangle tt \ (\phi \text{ propositional})$$

$$tr(\neg\varphi) = \neg tr(\varphi)$$

$$tr(\varphi_1 \wedge \varphi_2) = tr(\varphi_1) \wedge tr(\varphi_2)$$

$$tr(\varphi_1 \vee \varphi_2) = tr(\varphi_1) \vee tr(\varphi_2)$$

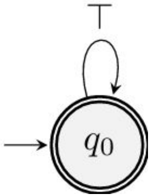
$$tr(\bigcirc\varphi) = \langle true\rangle(tr(\varphi) \wedge \neg end)$$

$$tr(\varphi_1 \,\mathcal{U}\, \varphi_2) = \langle(tr(\varphi_1)?; true)^*\rangle(tr(\varphi_2) \wedge \neg end)$$

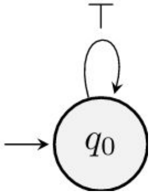
LDL$_f$ formula

DFA

$$tt$$



$$ff$$

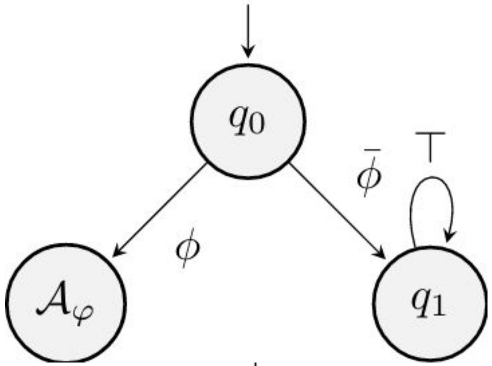


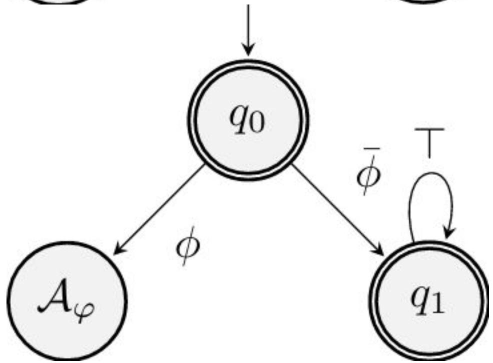
LDL$_f$ formula

$$\langle \phi \rangle \varphi$$

$$[\phi]\varphi$$

DFA





LDL$_f$ formula	DFA
$\varphi \wedge \psi$	$\mathcal{A}_\varphi \cap \mathcal{A}_\psi$
$\varphi \vee \psi$	$\mathcal{A}_\varphi \cup \mathcal{A}_\psi$
$\neg\varphi$	$\overline{\mathcal{A}_\varphi}$

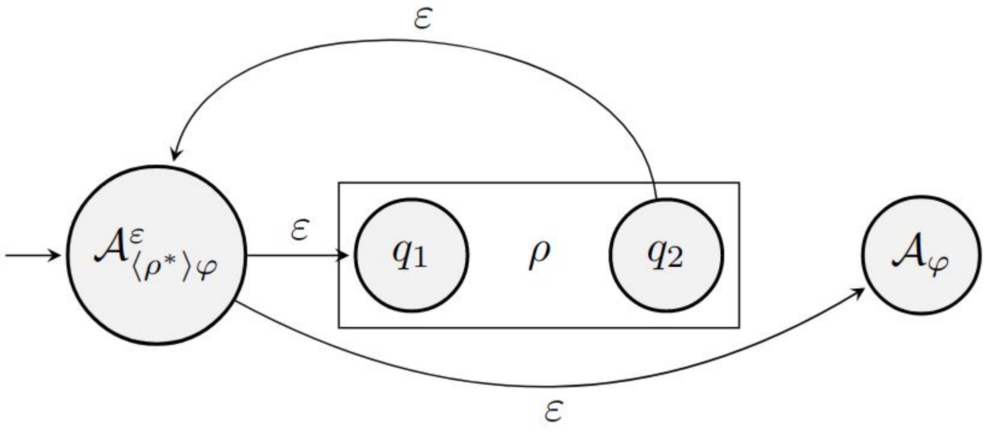
LDL_f formula

$$\langle \rho^* \rangle \varphi$$

DFA

1. Compute $\mathcal{A}_{\langle \rho \rangle end}$
2. Compute Kleene closure $\mathcal{A}^*_{\langle \rho \rangle end}$
3. Compute $\mathcal{A}_{\varphi}$
4. Concatenate $\mathcal{A}^*_{\langle \rho \rangle end}$ and $\mathcal{A}_{\varphi}$

$\mathcal{E}$-NFA equivalent to $\langle\rho^*\rangle\varphi$



(if ρ is test-free)

LDL$_f$ equivalences

$$\langle \psi ? \rangle \varphi \equiv \psi \wedge \varphi$$

$$[\psi ?] \varphi \equiv \neg \psi \vee \varphi$$

$$\langle \rho_1 ; \rho_2 \rangle \varphi \equiv \langle \rho_1 \rangle \langle \rho_2 \rangle \varphi$$

$$[\rho_1 ; \rho_2] \varphi \equiv [\rho_1][\rho_2] \varphi$$

$$\langle \rho_1 + \rho_2 \rangle \varphi \equiv \langle \rho_1 \rangle \psi \vee \langle \rho_2 \rangle \varphi$$

$$[\rho_1 + \rho_2] \varphi \equiv [\rho_1] \psi \wedge [\rho_2] \varphi$$

$$[\rho^*] \varphi \equiv \neg \langle \rho^* \rangle \neg \varphi$$

Example

Let $\varphi = \langle a + b \rangle \langle c; d \rangle tt$.
Transform it into:

$$\varphi' = \langle a \rangle \langle c \rangle \langle d \rangle tt \vee \langle b \rangle \langle c \rangle \langle d \rangle tt$$

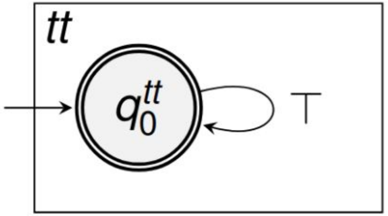
Note that $\varphi \equiv \varphi'$.

Example

$$\varphi' = \langle a \rangle \langle c \rangle \langle d \rangle tt \vee \langle b \rangle \langle c \rangle \langle d \rangle tt.$$

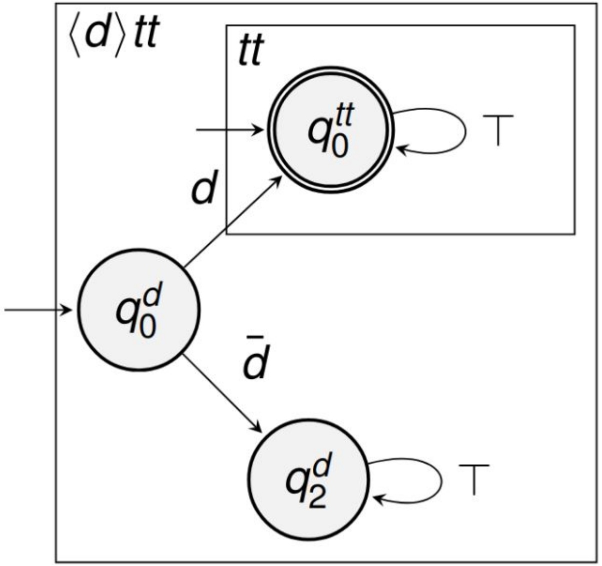
Example

$$\varphi' = \langle a \rangle \langle c \rangle \langle d \rangle tt \vee \langle b \rangle \langle c \rangle \langle d \rangle tt.$$



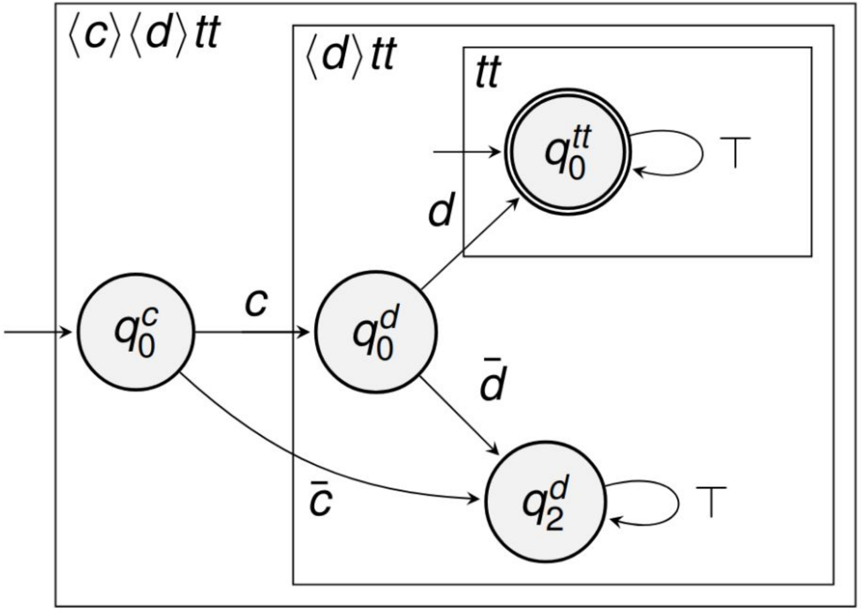
Example

$$\varphi' = \langle a \rangle \langle c \rangle \langle d \rangle tt \vee \langle b \rangle \langle c \rangle \langle d \rangle tt.$$



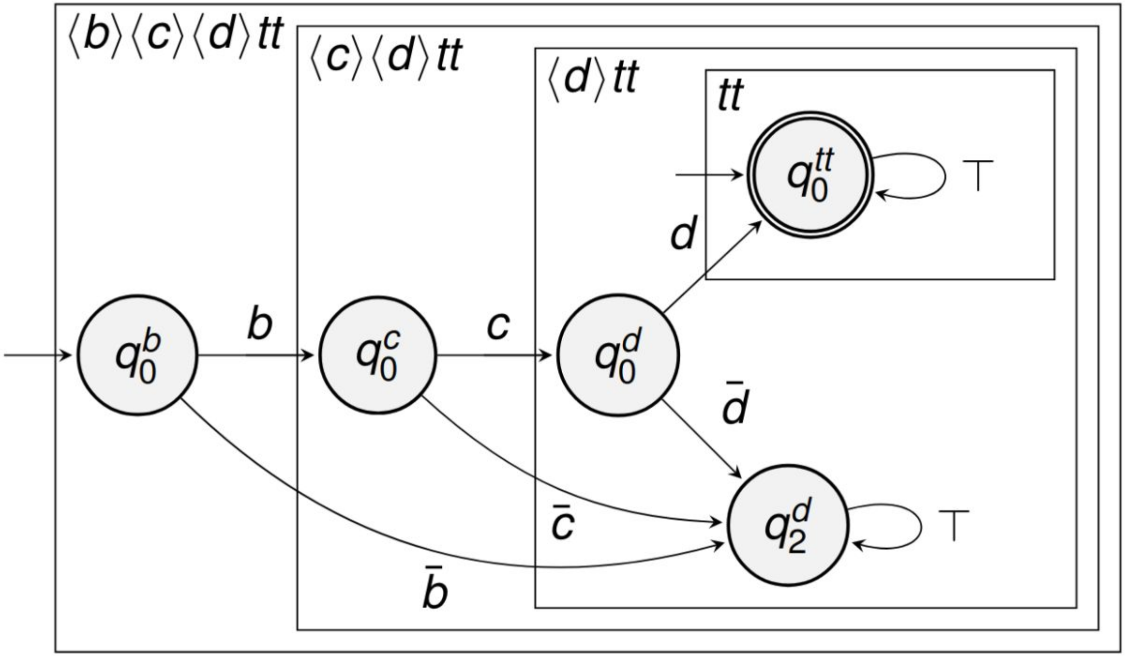
Example

$\varphi' = \langle a \rangle \langle c \rangle \langle d \rangle tt \vee \langle b \rangle \langle c \rangle \langle d \rangle tt.$



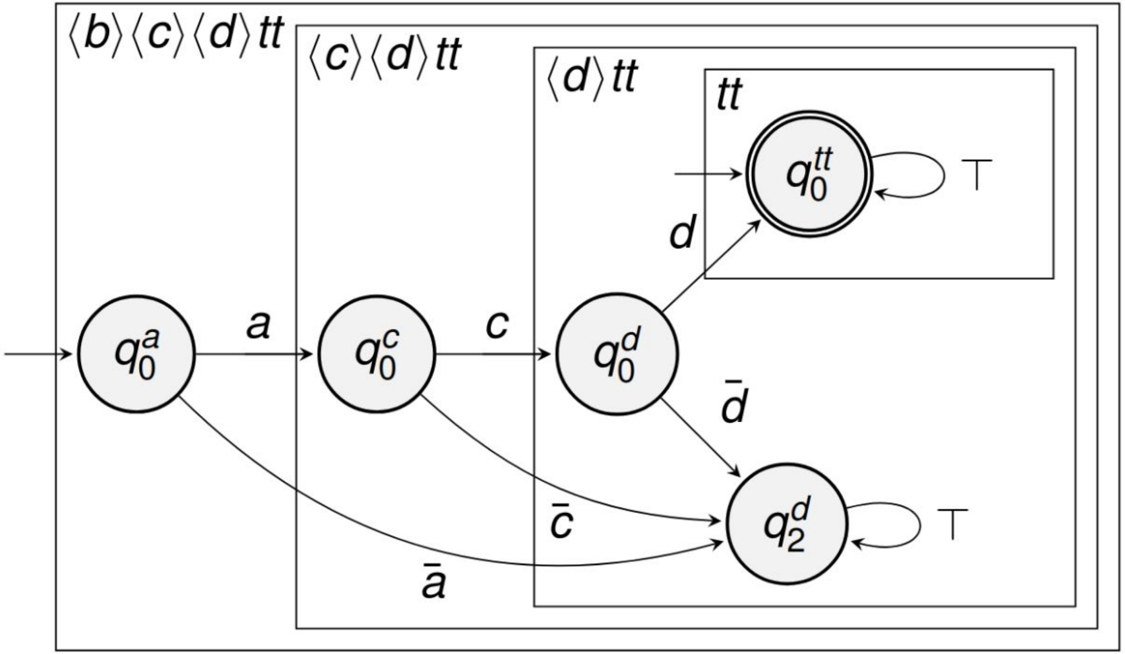
Example

$\varphi' = \langle a \rangle \langle c \rangle \langle d \rangle tt \vee \langle b \rangle \langle c \rangle \langle d \rangle tt.$



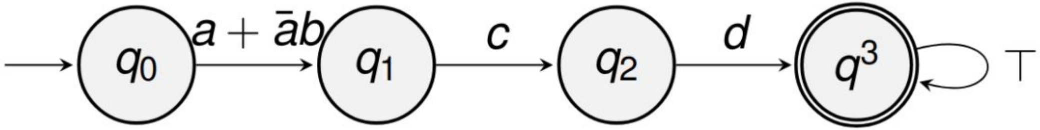
Example

$\varphi' = \langle a \rangle \langle c \rangle \langle d \rangle tt \vee \langle b \rangle \langle c \rangle \langle d \rangle tt$. (The same as before, but replacing b with a):



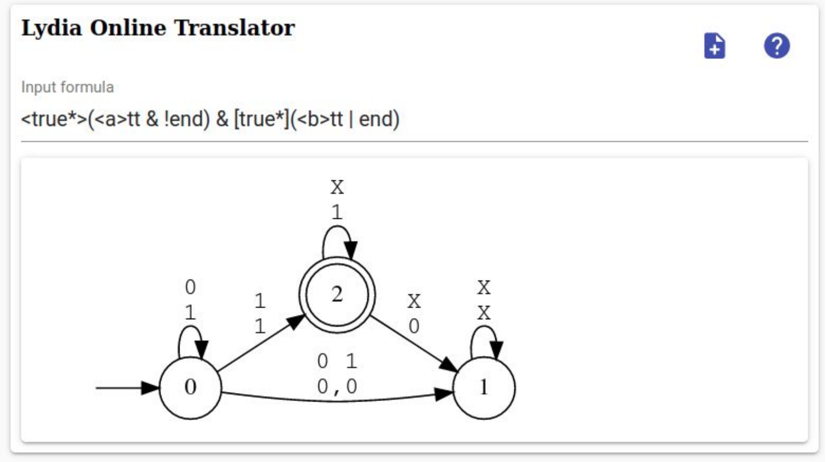
Example

Finally, $\mathcal{A}_{\varphi'} = \mathcal{A}_{\langle a \rangle \langle c \rangle \langle d \rangle tt} \cup \mathcal{A}_{\langle b \rangle \langle c \rangle \langle d \rangle tt}$.



Lydia Links

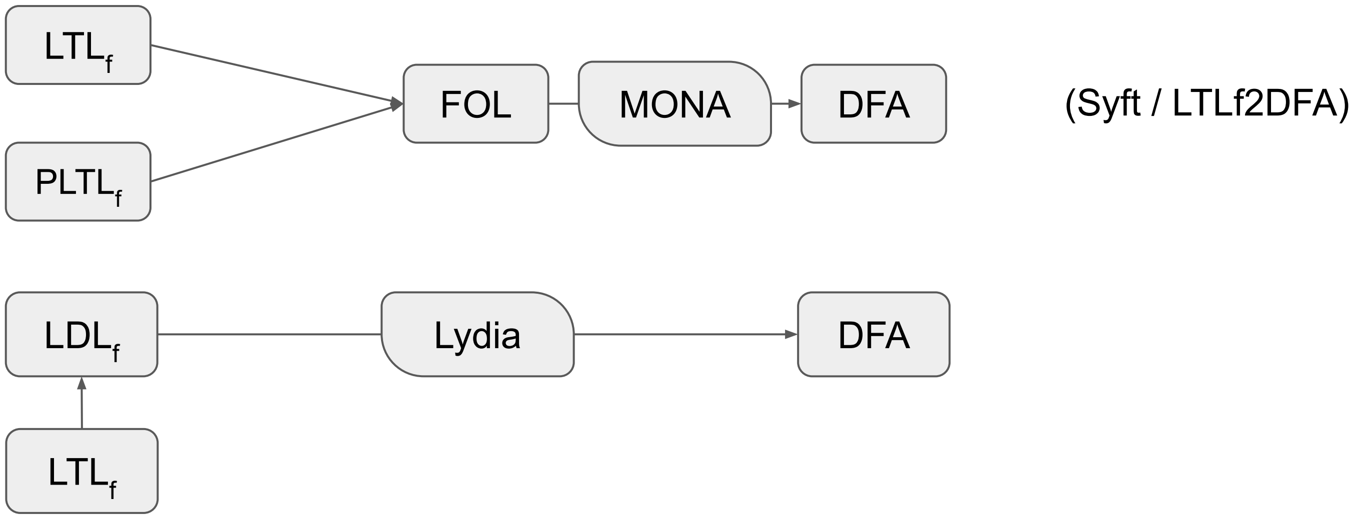
- [whitemech/lydia]
- [whitemech/lydia-benchmark]
- [whitemech/lydia-web-app]



https://lydia.whitemech.it/

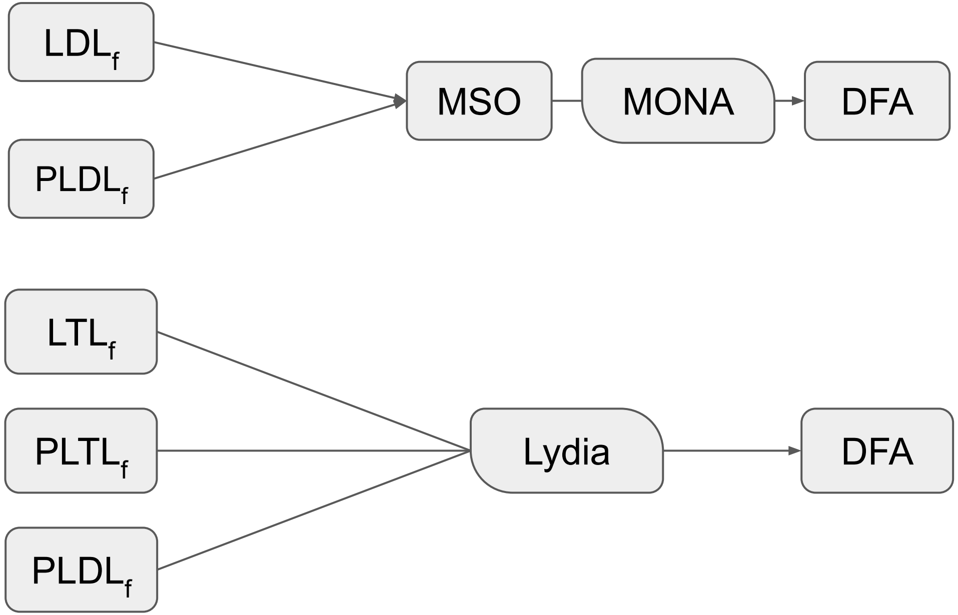
Takeaway

For your projects, you will most likely want to use one of the following:



Takeaway

What you could implement:



Python packages

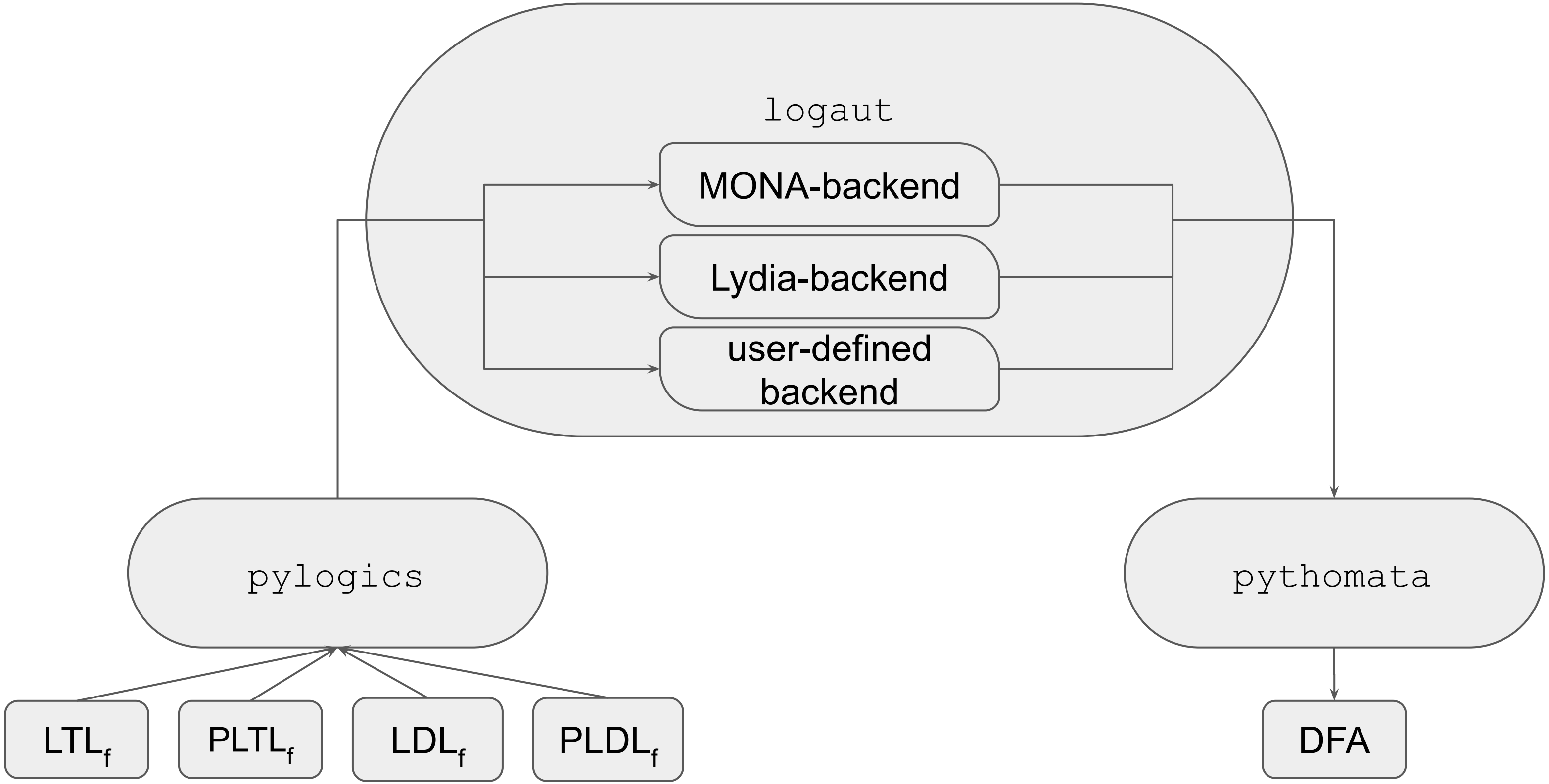
We implemented several Python packages to make it easy to use the above algorithms

whitemech/pylogics: library to handle temporal logic formulae

whitemech/pythomata: library to handle automata

whitemech/logaut: from temporal LOGics to AUTomata

- wrap tools like MONA and Lydia to translate logics into DFA


logaut
MONA-backend
Lydia-backend
user-defined backend
pylogics
pythomata
LTL$_f$
PLTL$_f$
LDL$_f$
PLDL$_f$
DFA

Pylogics

A Python library for logic formalisms representation and manipulation.

Code: https://github.com/whitemech/pylogics

Docs: https://whitemech.github.io/pylogics/

```
from pylogics.parsers import parse_pl
formula = parse_pl("(a & b) | (c & d)")

from pylogics.semantics.pl import evaluate_pl
evaluate_pl(formula, {'a'})  # returns False
evaluate_pl(formula, {'a', 'b'})  # returns True
```

Pylogics (LTL_f)

```
from pylogics.parsers import parse_ltl
parse_ltl("a")          # atom
parse_ltl("X(a)")       # next
parse_ltl("N(b)")       # weak next
parse_ltl("F(a)")       # eventually
parse_ltl("G(b)")       # always
parse_ltl("a U b")      # until
parse_ltl("a R b")      # release
parse_ltl("a W b")      # weak until
parse_ltl("a M b")      # strong release
```

Pylogics (PLTL$_f$)

```
from pylogics.parsers import parse_pltl
parse_pltl("Y(a)")    # before
parse_pltl("a S b")   # since
parse_pltl("O(b)")    # once
parse_pltl("H(a)")    # historically
```

Pylogics (LDL_f)

```
from pylogics.parsers import parse_ldl
parse_ldl("tt")
parse_ldl("ff")
parse_ldl("<a>tt")
parse_ldl("[a & b]ff")
parse_ldl("<a + b>tt")
parse_ldl("<a ; b><c>tt")
parse_ldl("<(a ; b)*><c>tt")
parse_ldl("<true><a>tt")  # Next a
parse_ldl("<(?<a>tt;true)*>(<b>tt)")  # (a Until b) in LDLf
```

Pylogics: supported features

Logics	Identifier	Parsing	Syntax	Semantics
Propositional Logic	`pl`	✓	✓	✓
Linear Temporal Logic (fin. traces)	`ltl`	✓	✓	✗
Past Linear Temporal Logic (fin. traces)	`pltl`	✓	✓	✗
Linear Dynamic Logic (fin. traces)	`ldl`	✓	✓	✗
Past Linear Dynamic Logic (fin. traces)	`pldl`	✗	✗	✗
First-order Logic	`fol`	✗	✗	✗
Monadic Second-order Logic	`mso`	✗	✗	✗

Pylogics: supported features

Logics	Identifier	Parsing	Syntax	Semantics
Propositional Logic	`pl`	✓	✓	✓
Linear Temporal Logic (fin. traces)	`ltl`	✓	✓	✗
Past Linear Temporal Logic (fin. traces)	`pltl`	✓	✓	✗
Linear Dynamic Logic (fin. traces)	`ldl`	✓	✓	✗
Past Linear Dynamic Logic (fin. traces)	`pldl`	✗	✗	✗
First-order Logic	`fol`	✗	✗	✗
Monadic Second-order Logic	`mso`	✗	✗	✗

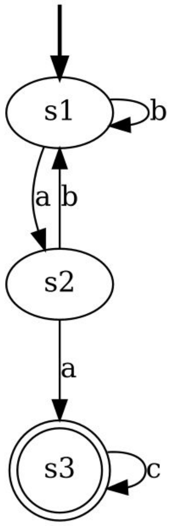
Projects!

Pythomata

Python library to handle automata:

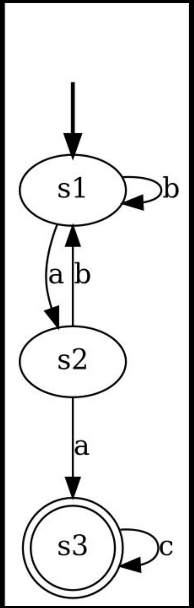
Code: https://github.com/whitemech/pythomata

Docs: https://whitemech.github.io/pythomata/



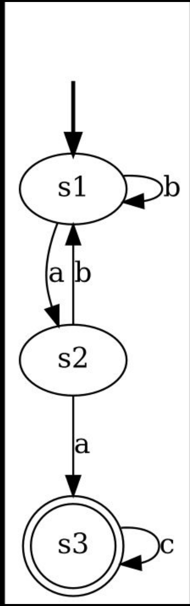
Pythomata example

```
from pythomata import SimpleDFA
alphabet = {"a", "b", "c"}
states = {"s1", "s2", "s3"}
initial_state = "s1"
accepting_states = {"s3"}
transition_function = {
    "s1": {
        "b" : "s1",
        "a" : "s2"
    },
    "s2": {
        "a" : "s3",
        "b" : "s1"
    },
    "s3":{
        "c" : "s3"
    }
}
dfa = SimpleDFA(states, alphabet, initial_state, accepting_states, transition_function)
```



Pythomata example

```
dfa.states
Out[3]: {'s1', 's2', 's3'}
dfa.initial_state
Out[4]: 's1'
dfa.accepting_states
Out[5]: {'s3'}
list(dfa.alphabet)
Out[6]: ['a', 'c', 'b']
dfa.transition_function
Out[7]: {
's1':
  {'b': 's1', 'a': 's2'},
's2':
  {'a': 's3', 'b': 's1'},
's3': {'c': 's3'}
}
```



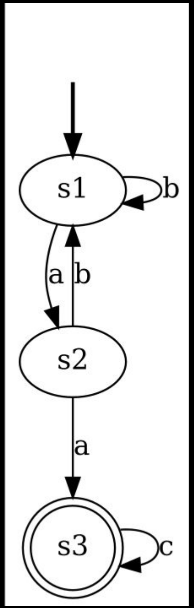
Pythomata example

```
# a word is a list of symbols
word = "bbbac"
dfa.accepts(word)            # True

# without the last symbol c,
# the final state is not reached
dfa.accepts(word[:-1])     # False

# operations
dfa_minimized = dfa.minimize()
dfa_trimmed = dfa.trim()

# print
dfa_trimmed.to_graphviz().render("path_to_file")
```



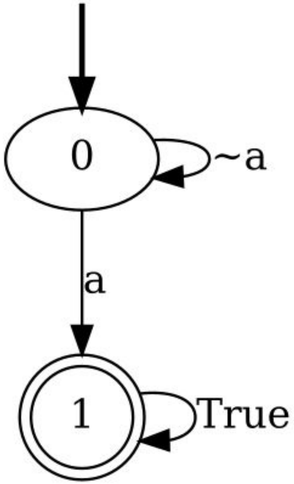
Pythomata example with Symbolic DFA

```
from pythomata.impl.symbolic import SymbolicDFA
automaton = SymbolicDFA()
q0 = 0
q1 = automaton.create_state()

automaton.set_initial_state(q0)
automaton.set_accepting_state(q1, True)

automaton.add_transition((q0, "~a", q0))
automaton.add_transition((q0, "a", q1))
automaton.add_transition((q1, "true", q1))

automaton.to_graphviz().render("dfa")
```



Logaut

- The “Keras” of temporal-logics-to-DFA
- You can extend it by implementing a custom backend
- Code: https://github.com/whitemech/logaut

```
from logaut import ltl2dfa
from pylogics.parsers import parse_ltl
formula = parse_ltl("F(a)")                    # pylogics' formula
dfa = ltl2dfa(formula, backend="lydia")  # pythomata's DFA
```

Logaut: custom Backend

```python
from logaut.backends.base import Backend

class MyBackend(Backend):

    def ltl2dfa(self, formula: Formula) -> DFA:

        """From LTL to DFA."""

    def ldl2dfa(self, formula: Formula) -> DFA:

        """From LDL to DFA."""

    def pltl2dfa(self, formula: Formula) -> DFA:

        """From PLTL to DFA."""

    def pldl2dfa(self, formula: Formula) -> DFA:

        """From PLDL to DFA."""

    def fol2dfa(self, formula: Formula) -> DFA:

        """From FOL to DFA."""

    def mso2dfa(self, formula: Formula) -> DFA:

        """From MSO to DFA."""
```

```python
from logaut.backends import register
register(
    id ="my_backend",
    entry_point="dotted.path.to.MyBackend"
)
dfa = ltl2dfa(formula, backend="my_backend")
```

Currently supported backends:

- Lydia (only LTL_f/LDL_f)
- LTLf2DFA (only LTL_f/$PLTL_f$)

Projects

- LTL_f -> DFA
- $PLTL_f$ -> DFA
- $PLDL_f$ -> DFA
- LDL_f -> MSO -> DFA (using MONA)
- $PLDL_f$ -> MSO -> DFA (using MONA)
- Extensive benchmark between tools: Lydia/MONA/Lisa/SPOT
- Implementation of small features to libraries: pylogics, pythomata, logaut etc.

Contact me for more information: favorito@diag.uniroma1.it

References

1. MONA User Manual: https://www.brics.dk/mona/mona14.pdf
2. G. De Giacomo and M. Vardi. "Linear temporal logic and linear dynamic logic on finite traces." In IJCAI, 2013.
3. R. Brafman, G. De Giacomo, and F. Patrizi. LTLf/LDLf non-markovian rewards. In AAAI, 2018.
4. S. Bansal, Y. Li, L. Tabajara, and M. Vardi. Hybrid compositional reasoning for reactive synthesis from finite-horizon specifications. In AAAI 2020.
5. G. De Giacomo and M. Favorito, “Compositional Approach to Translate LTLf/LDLf into Deterministic Finite Automata,” in ICAPS 2021 (to appear)