# Compositional $\text{LDL}_f$-to-DFA

Marco Favorito
PhD Student
favorito@diag.uniroma1.it

DIAG - Sapienza University of Rome

Joint work with Giuseppe De Giacomo
Accepted at ICAPS 2021

# The problem

Given an $\texttt{LDL}_f$ formula $\varphi$, compute a $\texttt{DFA}$ $\mathcal{A}$ such that:

$$\forall \pi . \pi \models \varphi \iff \pi \in \mathcal{L}(\mathcal{A})$$

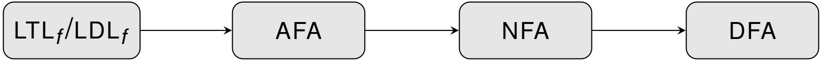
# Why care?

Basic building block of several techniques in AI and CS:

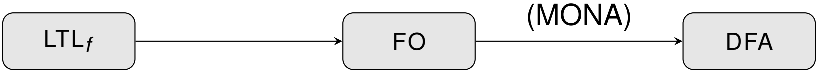
- Temporal Synthesis
- FOND Planning with temporal goals
- Non-Markovian Rewards Decision Processes
- Business Process Management

# Related work

(De Giacomo and Moshe Y. Vardi, 2013):




(Zhu et al., 2017; Bansal et al., 2020):




Our work:

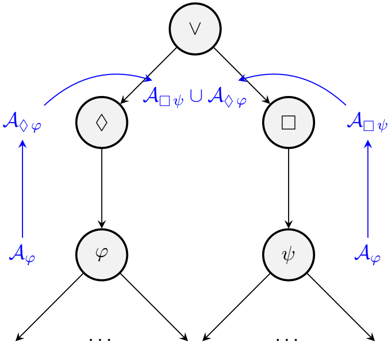
# A new technique

- Fully compositional
  - Like (Bansal et al., 2020), but to the extreme
- Bottom-up approach
  - Against “top-down” approach of AFA-NFA-DFA
- NONELEMENTARY (instead of best theoretical bound of 2EXPTIME)
  - Yet, it works fairly well in practice
  - MONA too is NONELEMENTARY!

# How it works, in a nutshell

- Mapping from LDL$_f$ operators to DFA operations
- *Inductively* apply these mappings
- If we encounter LTL$_f$ formulae, translate them in LDL$_f$

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



We use the syntax that also works for empty traces (Brafman, De Giacomo, and Patrizi, 2018).

Given a set of propositional symbols $\mathcal{P}$, $\text{LDL}_f$ formulae are built as follows:

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

Where $\phi$ is a propositional formula over $\mathcal{P}$.

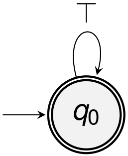
# $\text{LDL}_f$ captures $\text{LTL}_f$

The function *tr* encodes $\text{LTL}_f$ into $\text{LDL}_f$:

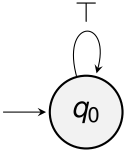
$$
\begin{aligned}
tr(\phi) &= \langle \phi \rangle tt \text{ } (\phi \text{ propositional}) \\
tr(\neg\varphi) &= \neg tr(\varphi) \\
tr(\varphi_1 \wedge \varphi_2) &= tr(\varphi_1) \wedge tr(\varphi_2) \\
tr(\varphi_1 \vee \varphi_2) &= tr(\varphi_1) \vee tr(\varphi_2) \\
tr(\circ\varphi) &= \langle true \rangle (tr(\varphi) \wedge \neg end) \\
tr(\varphi_1 \, \mathcal{U} \, \varphi_2) &= \langle (tr(\varphi_1)?; true)^* \rangle (tr(\varphi_2) \wedge \neg end)
\end{aligned}
$$

# Mappings from LDL$_f$ to DFA

*tt* $\longrightarrow$

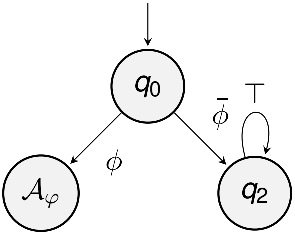



*ff* $\longrightarrow$

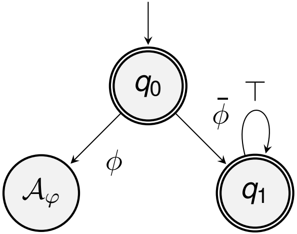


# Mappings from LDL$_f$ to DFA (cont.)

$\langle\rho\rangle\varphi \longrightarrow$




$[\rho]\varphi \longrightarrow$



# Mappings from LDL$_f$ to DFA (cont.)

$$\varphi \wedge \psi \longrightarrow \mathcal{A}_\varphi \cap \mathcal{A}_\psi$$

$$\varphi \vee \psi \longrightarrow \mathcal{A}_\varphi \cup \mathcal{A}_\psi$$

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

# LDL$_f$ equivalences

For other operators (except $\langle\rho^*\rangle\varphi$) we can exploit the following equivalences:

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

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

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

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

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

$$[\rho_1 + \rho_2]\varphi \equiv [\rho_1] \wedge \langle\rho_2\rangle\varphi$$

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

# Translation of $\langle \rho^* \rangle \varphi$

For the transformation of $\langle \rho^* \rangle \varphi$, we distinguish two cases:

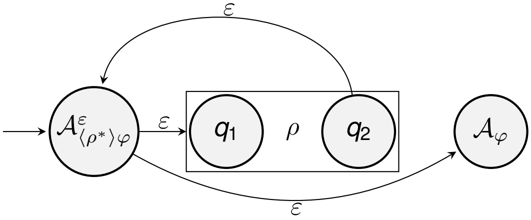
- $\rho$ is test-free;
- $\rho$ is *not* test-free

# Translation of $\langle \rho^* \rangle \varphi$ ($\rho$ test-free)

If $\rho$ is test-free, then $\rho \equiv \langle \rho \rangle \mathit{end}$.
To obtain $\mathcal{A}_{\langle \rho^* \rangle \varphi}$:

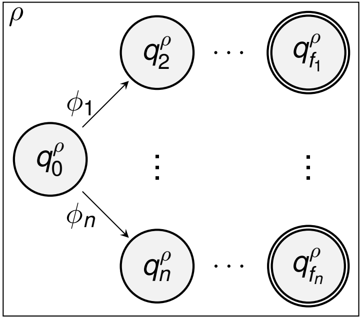
- Compute $\mathcal{A}_{\langle \rho \rangle \mathit{end}}$
- Compute the Kleene closure of $\mathcal{A}_{\langle \rho \rangle \mathit{end}}$, $\mathcal{A}_{\rho^*}$
- Compute $\mathcal{A}_{\varphi}$
- Concatenate $\mathcal{A}_{\rho^*}$ and $\mathcal{A}_{\varphi}$

$\varepsilon$-NFA associated to $\langle \rho^* \rangle \varphi$:



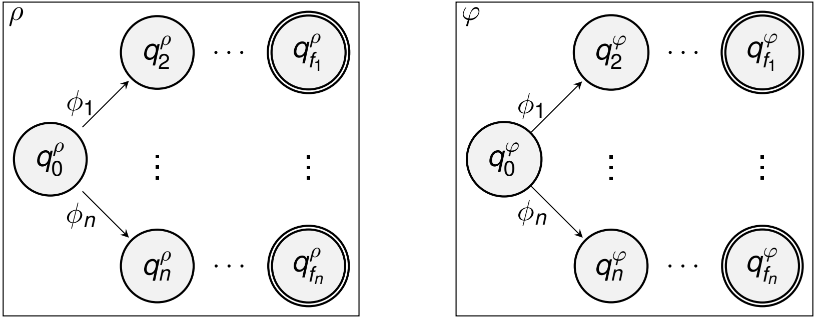
# Translation of $\langle \rho^* \rangle \varphi$ ($\rho$ test-free), with NFAs

- Compute $\mathcal{A}_{\langle \rho \rangle end}$:



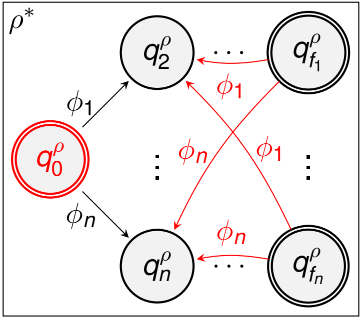
# Translation of $\langle \rho^* \rangle \varphi$ ($\rho$ test-free), with NFAs

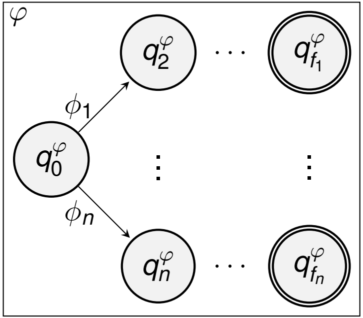
- Compute $\mathcal{A}_\varphi$:



# Translation of $\langle \rho^* \rangle \varphi$ ($\rho$ test-free), with NFAs

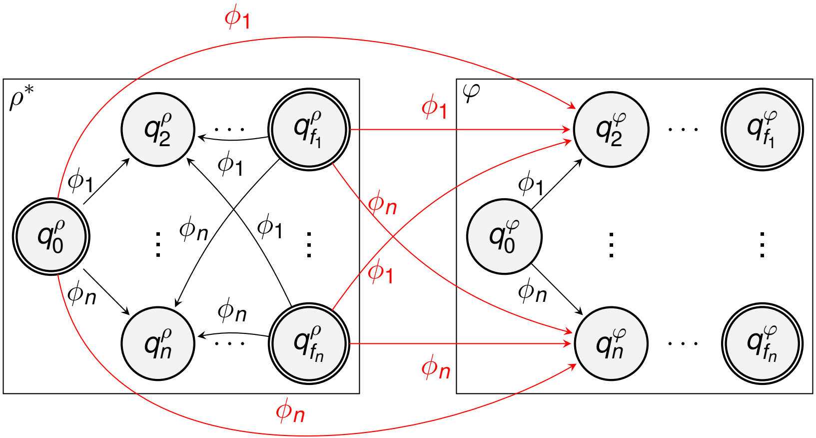
- Compute the Kleene closure of $\mathcal{A}_{\langle \rho \rangle}$ *end*, $\mathcal{A}_{\rho^*}$:



# Translation of $\langle \rho^* \rangle \varphi$ ($\rho$ test-free), with NFAs

- Concatenate $\mathcal{A}_{\rho^*}$ and $\mathcal{A}_{\varphi}$:



# Translation of $\langle \rho^* \rangle \varphi$ ($\rho$ not test-free)

If $\rho$ contains a test expression, we resort to the LDL$_f$-to-AFA transformation (De Giacomo and Moshe Y. Vardi, 2013; Brafman, De Giacomo, and Patrizi, 2018), with some changes:

- Pre-compute DFAs of tests $\psi_1?, \ldots, \psi_n?$ and $\varphi$;
- Instead of expanding states of the form $\psi_i?$ (or $\varphi$), concatenate the current state to the initial state of $\mathcal{A}_{\psi_i}$ (or $\mathcal{A}_{\varphi}$).

# Analysis

**Theorem (Correctness)**

*The presented technique is correct, i.e. it outputs a* DFA $\mathcal{A}_\varphi$ *s.t.*
$\forall \pi . \pi \models \varphi \iff \pi \in \mathcal{L}(\mathcal{A}_\varphi)$

**Proof.**

By structural induction on the formulae constructs $\varphi$, by structural induction on the regular expression constructs $\rho$, and by induction on the length of the trace $\pi$. □

Time complexity is NONELEMENTARY, because of arbitrary nested star operators.

# Implementation

# Lydia

The technique has been implemented in a tool called **Lydia**:

- It relies on **MONA** (Henriksen et al., 1995) for DFA representation and operations;
- It is integrated with **Syft+** for $\text{LTL}_f$/$\text{LDL}_f$ synthesis;
- Uses CUDD to find minimal models;
- It is able to parse both $\text{LDL}_f$ and $\text{LTL}_f$ formulae using Flex/Bison.

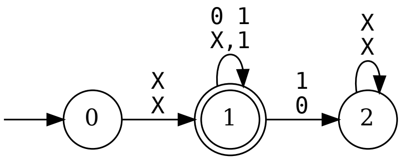
# The MONA DFA library

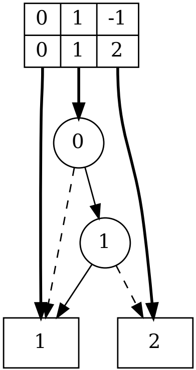
MONA is a tool for translating Weak monadic Second-order theory of 1 Successor (WS1S) to DFAs.

Lydia *only* uses the MONA DFA library.

DFAs in MONA are represented by shared, multi-terminal BDDs.

The representation is *explicit* in the state space, and *symbolic* in the transitions.



# Alternation in MONA

Problem: the MONA DFA library cannot represent NFAs or AFAs directly

... but it provides the (existential) projection operation, EPROJECT($\mathcal{A}$, $i$):

- remove the $i$th track from the MBDD of $\mathcal{A}$;
- determinize (as if it was a NFA)

# Alternation in MONA

Problem: the MONA DFA library cannot represent NFAs or AFAs directly

... but it provides the (existential) projection operation, EPROJECT($\mathcal{A}$, *i*):

- remove the *i*th track from the MBDD of $\mathcal{A}$;
- determinize (as if it was a NFA)

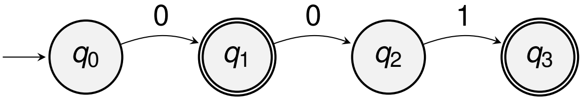
We also added the *universal* projection operation, UPROJECT($\mathcal{A}$, *i*):

- remove the *i*th track from the MBDD of $\mathcal{A}$ (as above);
- determinize (as if it was an UFA (Universal Finite Automaton)).

# Kleene closure using EPROJECT (Yu et al., 2008)

$L = \{0, 001\}$, $\Sigma = \{0, 1\} = 2^{\{b\}}$ (only one bit $b$ needed)

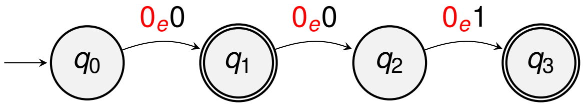
How to compute $L^*$?



# Kleene closure using EPROJECT (Yu et al., 2008)

$L = \{0, 001\}$, $\Sigma = \{0, 1\} = 2^{\{b\}}$ (only one bit $b$ needed)
How to compute $L^*$?

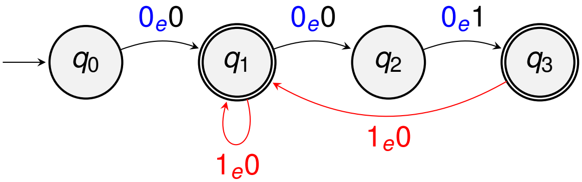
- Add existential bit "$e$";
- Set it to false (i.e. $\bar{e}$) to existing transitions.



# Kleene closure using EPROJECT (Yu et al., 2008)

$L = \{0, 001\}$, $\Sigma = \{0, 1\} = 2^{\{b\}}$ (only one bit $b$ needed)
How to compute $L^*$?

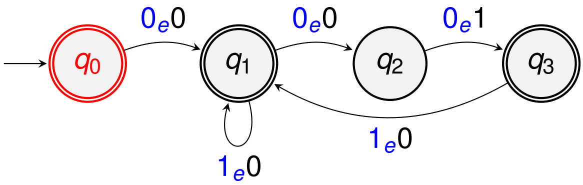
- Add closure transitions with bit $e$ set to true



# Kleene closure using EPROJECT (Yu et al., 2008)

$L = \{0, 001\}$, $\Sigma = \{0, 1\} = 2^{\{b\}}$ (only one bit $b$ needed)
How to compute $L^*$?

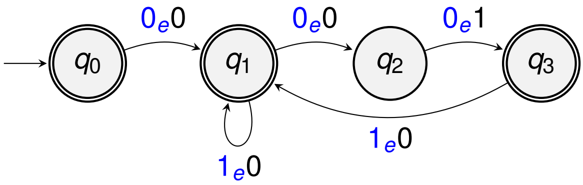
- Make initial state accepting



# Kleene closure using EPROJECT (Yu et al., 2008)

$L = \{0, 001\}$, $\Sigma = \{0, 1\} = 2^{\{b\}}$ (only one bit $b$ needed)
How to compute $L^*$?

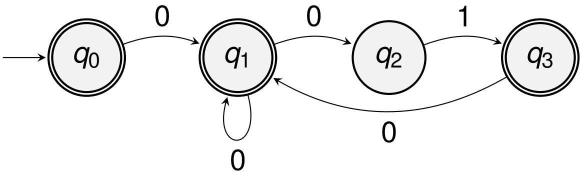
- EPROJECT($\mathcal{A}$, $i_e$) (project away bit $e$)



# Kleene closure using EPROJECT (Yu et al., 2008)

$L = \{0, 001\}$, $\Sigma = \{0, 1\} = 2^{\{b\}}$ (only one bit $b$ needed)
How to compute $L^*$?

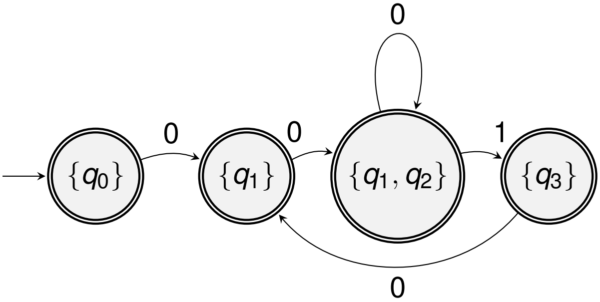
- EPROJECT($\mathcal{A}$, $i_e$) (project away bit $e$)



# Kleene closure using EPROJECT (Yu et al., 2008)

$L = \{0, 001\}$, $\Sigma = \{0, 1\} = 2^{\{b\}}$ (only one bit $b$ needed)
How to compute $L^*$?

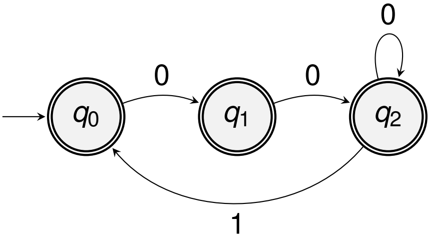
- EPROJECT($\mathcal{A}$, $i_e$) (determinize)



# Kleene closure using EPROJECT (Yu et al., 2008)

$L = \{0, 001\}$, $\Sigma = \{0, 1\} = 2^{\{b\}}$ (only one bit $b$ needed)
How to compute $L^*$?

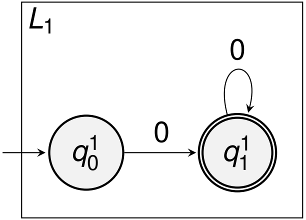
- Minimize

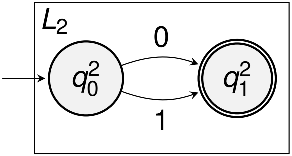


# Concatenation using EPROJECT (Yu et al., 2008)

$L_1 = \{00^*\}, L_2 = \{0 + 1\}$

How to compute $L_1 L_2$?

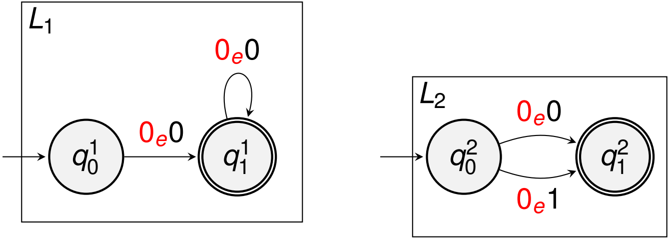


# Concatenation using EPROJECT (Yu et al., 2008)

$L_1 = \{00^*\}, L_2 = \{0+1\}$
How to compute $L_1 L_2$?

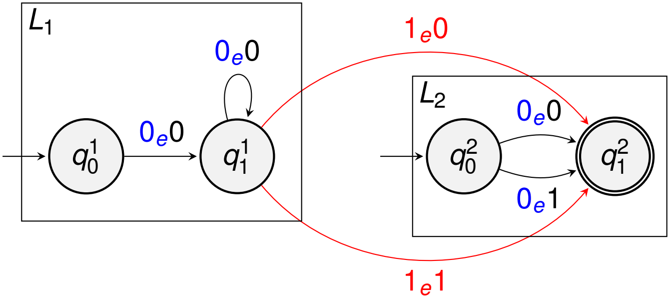
- Add existential bit "$e$";
- Set it to false (i.e. $\bar{e}$) to existing transitions.



# Concatenation using EPROJECT (Yu et al., 2008)

$L_1 = \{00^*\}, L_2 = \{0+1\}$
How to compute $L_1 L_2$?

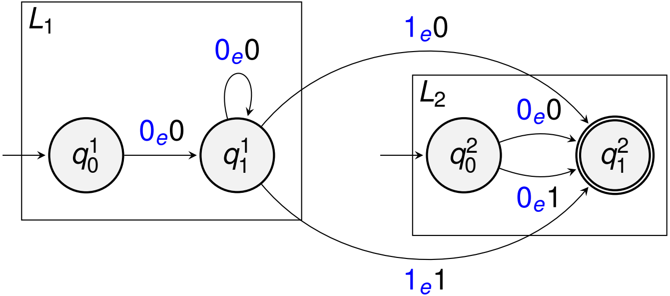
- Add concatenation transitions with bit $e$ set to true.



# Concatenation using EPROJECT (Yu et al., 2008)

$L_1 = \{00^*\}, L_2 = \{0 + 1\}$
How to compute $L_1 L_2$?

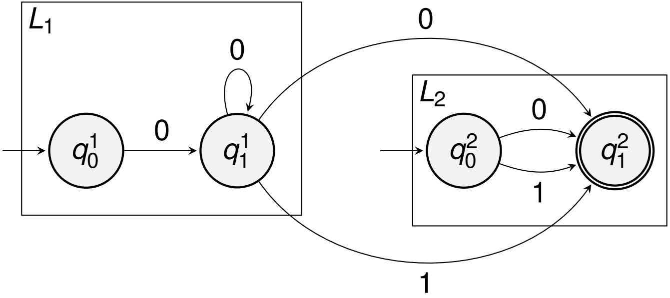
- EPROJECT($\mathcal{A}$, $i_e$) (project away bit $e$)



# Concatenation using EPROJECT (Yu et al., 2008)

$L_1 = \{00^*\}, L_2 = \{0 + 1\}$
How to compute $L_1 L_2$?

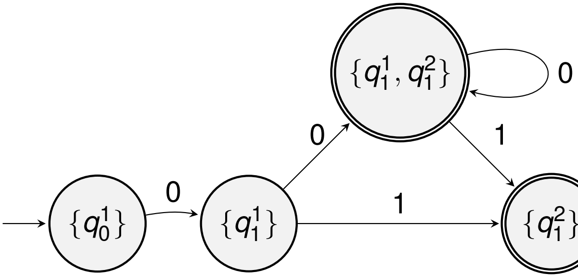
- EPROJECT($\mathcal{A}$, $i_e$) (project away bit $e$)



# Concatenation using EPROJECT (Yu et al., 2008)

$L_1 = \{00^*\}, L_2 = \{0 + 1\}$
How to compute $L_1 L_2$?

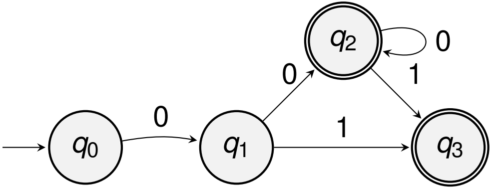
- EPROJECT($\mathcal{A}$, $i_e$) (determinize)



# Concatenation using EPROJECT (Yu et al., 2008)

$L_1 = \{00^*\}, L_2 = \{0 + 1\}$
How to compute $L_1 L_2$?

- Minimize



# Example for $\varphi$ (no star operators)

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

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

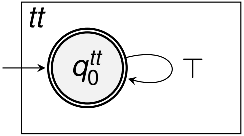
Note that $\varphi \equiv \varphi'$.

# Example for $\varphi$ (no star operators)

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

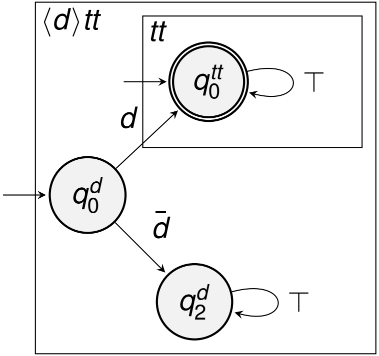
# Example for $\varphi$ (no star operators)

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



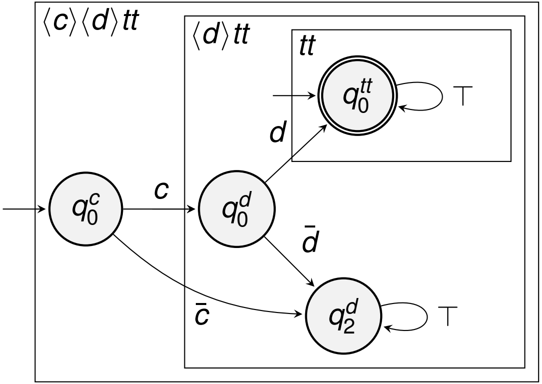
# Example for $\varphi$ (no star operators)

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



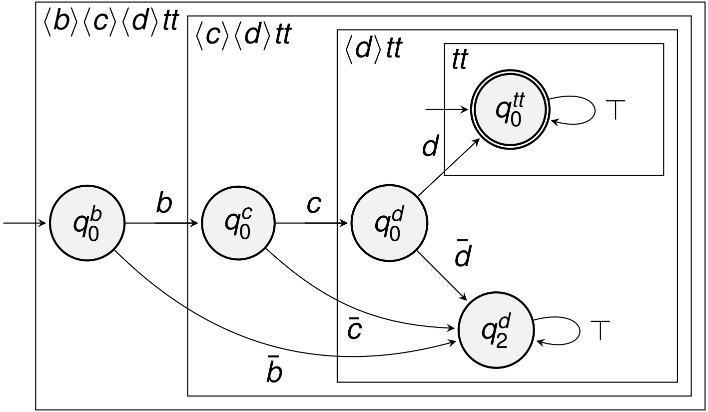
# Example for $\varphi$ (no star operators)

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



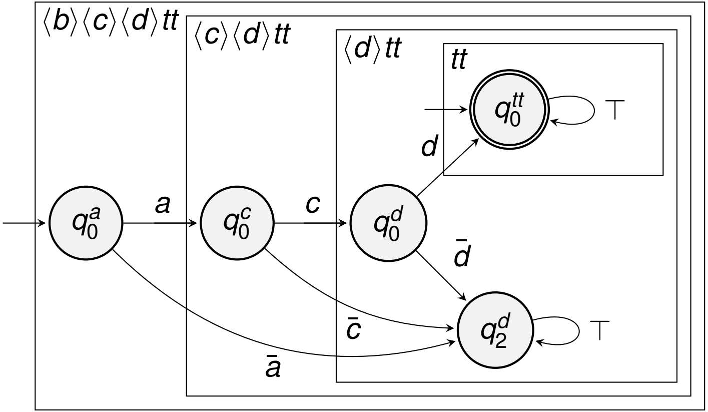
# Example for $\varphi$ (no star operators)

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



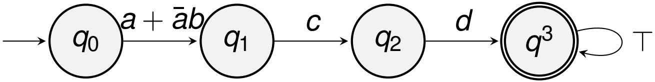
# Example for $\varphi$ (no star operators)

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



# Example for $\varphi$ (no star operators)

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



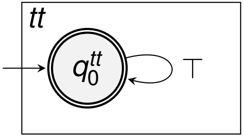
# Example for $\langle\rho^*\rangle\varphi$ (test free)

Let $\varphi = [a^*]\langle b\rangle tt$

# Example for $\langle \rho^* \rangle \varphi$ (test free)

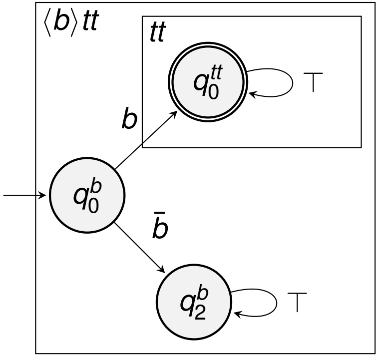
Let $\varphi = [a^*]\langle b \rangle tt$

- Compute $\mathcal{A}_{tt}$



Let $\varphi = [a^*]\langle b \rangle tt$

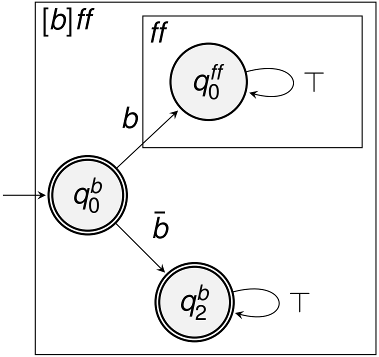
- Compute $\mathcal{A}_{\langle b \rangle tt}$



# Example for $\langle \rho^* \rangle \varphi$ (test free)

Let $\varphi = [a^*]\langle b \rangle tt$ (remember: $[\rho]\varphi \equiv \neg\langle\rho\rangle\neg\varphi$)

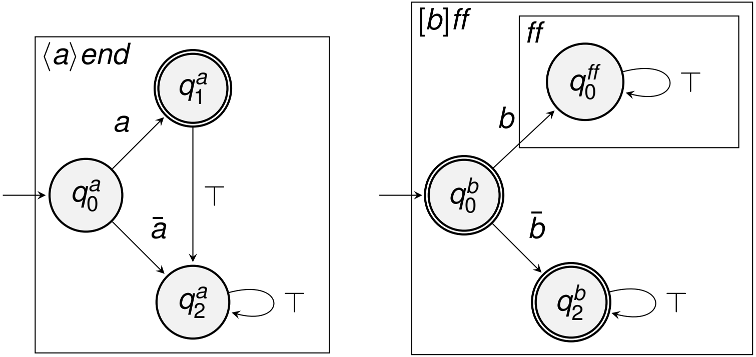
- Compute $\overline{\mathcal{A}_{\langle b \rangle tt}} = \mathcal{A}_{[b] ff}$



# Example for $\langle \rho^* \rangle \varphi$ (test free)

Let $\varphi = [a^*]\langle b \rangle tt$

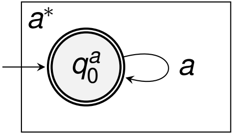
- Compute $\mathcal{A}_{\langle a \rangle end}$

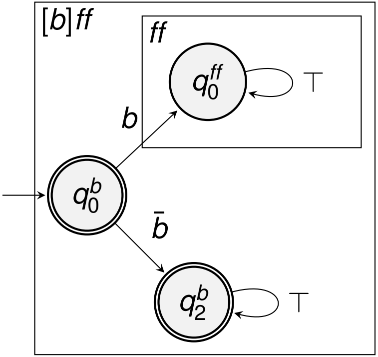


# Example for $\langle \rho^* \rangle \varphi$ (test free)

Let $\varphi = [a^*]\langle b \rangle tt$

- Compute Kleene closure of $\mathcal{A}_{\langle a \rangle end}$, $\mathcal{A}_{a^*}$

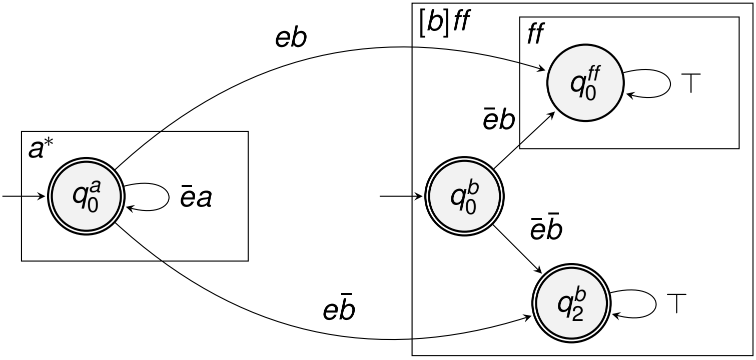


# Example for $\langle \rho^* \rangle \varphi$ (test free)

Let $\varphi = [a^*]\langle b \rangle tt$

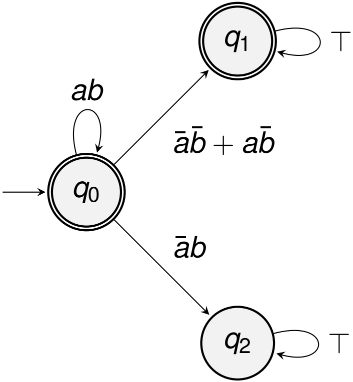
- Concatenate $\mathcal{A}_{a^*}$ and $\mathcal{A}_{[b]ff}$ (note: $\bar{e}a \wedge eb = \bot$)



# Example for $\langle \rho^* \rangle \varphi$ (test free)

Let $\varphi = [a^*]\langle b \rangle tt$

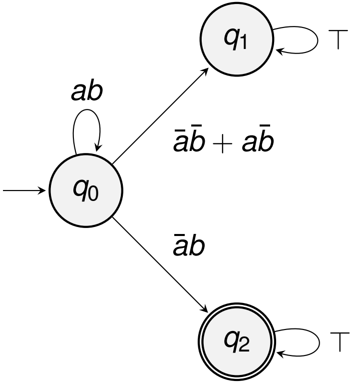
- Do EPROJECT($\mathcal{A}'_{\langle a^* \rangle [b] ff}$, $i_e$)



# Example for $\langle \rho^* \rangle \varphi$ (test free)

Let $\varphi = [a^*]\langle b \rangle tt$

- $\overline{\mathcal{A}_{\langle a^* \rangle [b] ff}} = \mathcal{A}_{[a^*]\langle b \rangle tt}$



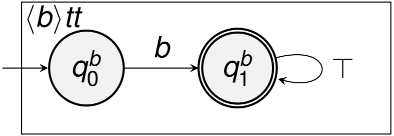
# Example for $\langle \rho^* \rangle \psi$ (with tests)

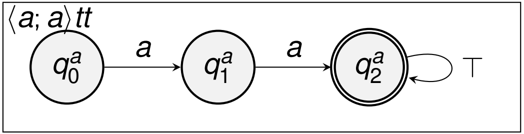
Let $\varphi = \langle (\langle a; a \rangle tt?; true)^* \rangle \langle b \rangle tt$
(Note: $\langle a; a \rangle tt$ requires two steps to be verified.)
We need to compute its AFA.

# Example for $\langle \rho^* \rangle \psi$ (with tests)

Let $\varphi = \langle (\langle a; a \rangle tt?; true)^* \rangle \langle b \rangle tt$

- Precompute $\mathcal{A}_{\langle a;a \rangle tt}$ and $\langle b \rangle tt$



# Example for $\langle \rho^* \rangle \psi$ (with tests)

Let $\varphi = \langle (\langle a; a \rangle tt?; true)^* \rangle \langle b \rangle tt$

- Start from $q_0 = \varphi$

# Example for $\langle \rho^* \rangle \psi$ (with tests)

Let $\varphi = \langle ((\langle a; a \rangle tt?; true)^* \rangle \langle b \rangle tt$

- Start from $q_0 = \varphi$
- "Expand" $q_0$, without consuming symbols:

$$\tilde{\delta}(\varphi) = \text{``}\text{``}\langle b \rangle tt\text{''}\text{''} \vee (\text{``}\langle a; a \rangle tt?\text{''} \wedge \text{``}\langle true \rangle \boldsymbol{F}_\varphi\text{''})$$

(Note: ""$\langle b \rangle tt$"" is double-quoted)

# Example for $\langle \rho^* \rangle \psi$ (with tests)

Let $\varphi = \langle (\langle a; a \rangle tt?; true)^* \rangle \langle b \rangle tt$

- Start from $q_0 = \varphi$
- "Expand" $q_0$, without consuming symbols:

$$\tilde{\delta}(\varphi) = \text{``“}\langle b \rangle tt\text{”''} \vee (\text{``}\langle a; a \rangle tt?\text{''} \wedge \text{``}\langle true \rangle \mathbf{F}_{\varphi}\text{''})$$

(Note: ``“$\langle b \rangle tt$”'' is double-quoted)

- the above formula will determine the next transitions from the current state.

# Example for $\langle \rho^* \rangle \psi$ (with tests)

$$\tilde{\delta}(\varphi) = \text{``}\text{``}\langle b \rangle tt\text{''}\text{''} \vee (\text{``}\langle a; a \rangle tt?\text{''} \wedge \text{``}\langle true \rangle \boldsymbol{F}_\varphi\text{''})$$

# Example for $\langle\rho^*\rangle\psi$ (with tests)

$$\tilde{\delta}(\varphi) = \text{“}\text{“}\langle b\rangle tt\text{”}\text{”} \vee (\text{“}\langle a; a\rangle tt?\text{”} \wedge \text{“}\langle true\rangle \boldsymbol{F}_{\varphi}\text{”})$$

Compute minimal models of the above (propositional) formula:

1. $\{\text{“}\langle b\rangle tt\text{”}\text{”}\}$
2. $\{\text{“}\langle a; a\rangle tt?\text{”}, \text{“}\langle true\rangle \boldsymbol{F}_{\varphi}\text{”}\}$

# Example for $\langle \rho^* \rangle \psi$ (with tests)

$$\tilde{\delta}(\varphi) = \text{``}\text{``}\langle b \rangle tt\text{''}\text{''} \vee (\text{``}\langle a; a \rangle tt?\text{''} \wedge \text{``}\langle true \rangle \boldsymbol{F}_\varphi\text{''})$$

Compute minimal models of the above (propositional) formula:

1. $\{\text{``}\langle b \rangle tt\text{''}\text{''}\}$
2. $\{\text{``}\langle a; a \rangle tt?\text{''}, \text{``}\langle true \rangle \boldsymbol{F}_\varphi\text{''}\}$

Intuitively, it means:

- take initial transitions from $\mathcal{A}_{\langle b \rangle tt}$, ***or***
- take initial transitions from $\mathcal{A}_{\langle a;a \rangle tt}$, ***and***, go to $\boldsymbol{E}(\varphi)$ if you read *true*

(Note, $\boldsymbol{E}(\varphi)$ is an AFA state)

# Example for $\langle \rho^* \rangle \psi$ (with tests)

**Remark:** Add as many bits as needed, existential $e_i$ and universal $u_i$, to resolve all the alternations, i.e. to have only deterministic transitions.

In the example, we need one existential bit $e$ and one universal bit $u$.

**Remark:** Add as many bits as needed, existential $e_i$ and universal $u_i$, to resolve all the alternations, i.e. to have only deterministic transitions.

In the example, we need one existential bit $e$ and one universal bit $u$.

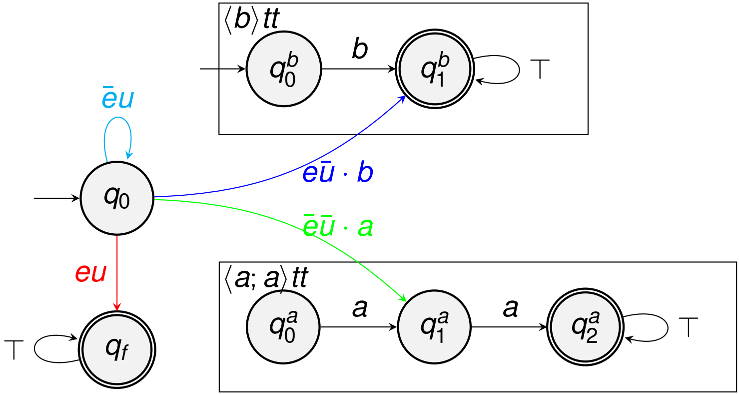
$$\tilde{\delta}(\varphi) = (\underset{e\bar{u}}{\text{““}\langle b\rangle tt\text{””}} \wedge \underset{eu}{true}) \vee (\underset{\bar{e}\bar{u}}{\text{“}\langle a; a\rangle tt?\text{”}} \wedge \underset{\bar{e}u}{\text{“}\langle true\rangle \mathbf{F}_{\varphi}\text{”}})$$

From $q_0$ (the current state in this iteration):

- $\bar{e}\bar{u}$: take *all* transitions from initial state of $\mathcal{A}_{\langle a;a\rangle tt}$;
- $\bar{e}u$: go to $\varphi = q_0$ (a self-loop)
- $e\bar{u}$: take *all* transitions from initial state of $\mathcal{A}_{\langle b\rangle tt}$;
- $eu$: go to accepting sink (requires “rebalancing” of the DNF formula)

# Example for $\langle \rho^* \rangle \psi$ (with tests)

$$(\text{“““}\underbrace{\langle b \rangle tt}_{e\bar{u}}\text{”””} \wedge \underbrace{true}_{eu}) \vee (\text{“}\underbrace{\langle a;a \rangle tt?}_{\bar{e}\bar{u}}\text{”} \wedge \text{“}\underbrace{\langle true \rangle \boldsymbol{F}_\varphi}_{\bar{e}u}\text{”})$$



# Example for $\langle \rho^* \rangle \psi$ (with tests)

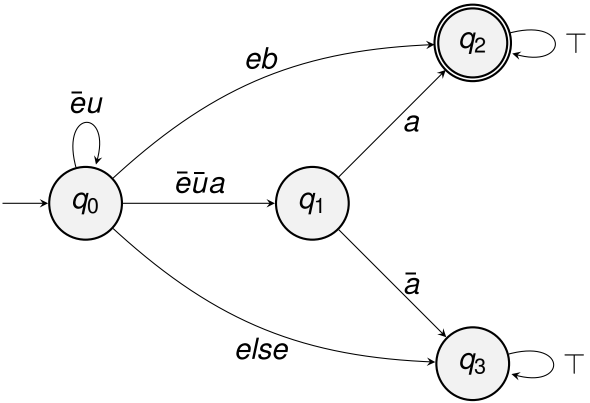
Iterate the above procedure until all AFA states have been explored.

To obtain the final DFA:

1. UPROJECT the universal bits;
2. EPROJECT the existential bits.

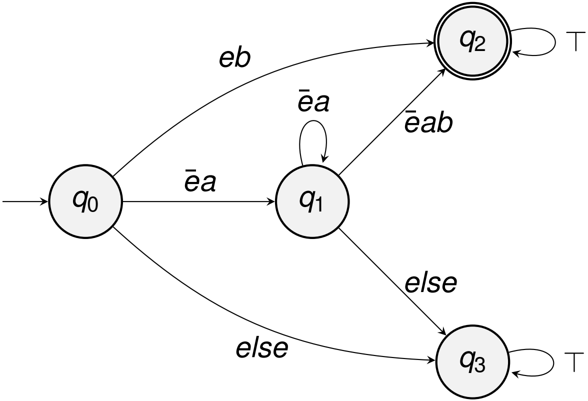
# Example for $\langle\rho^*\rangle\psi$ (with tests)

The above DFA, minimized:



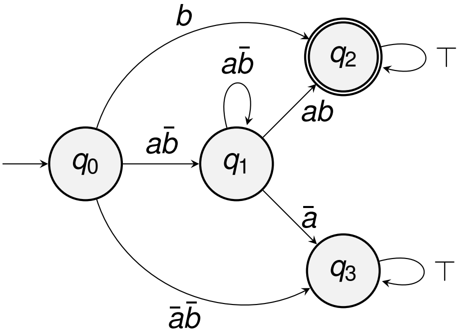
# Example for $\langle\rho^*\rangle\psi$ (with tests)

After UPROJECT:



# Example for $\langle \rho^* \rangle \psi$ (with tests)

After EPROJECT, the minimal DFA for $\varphi = \langle (\langle a; a \rangle tt?; true)^* \rangle \langle b \rangle tt$:



# Experiments

# Benchmark

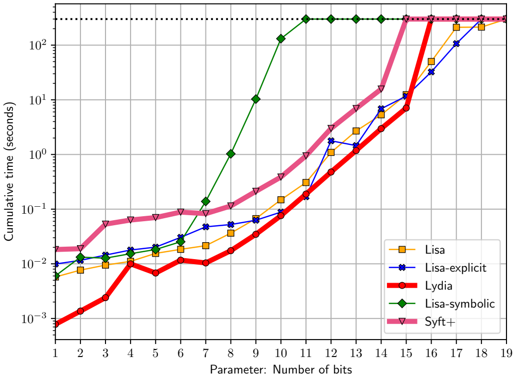
Tools:

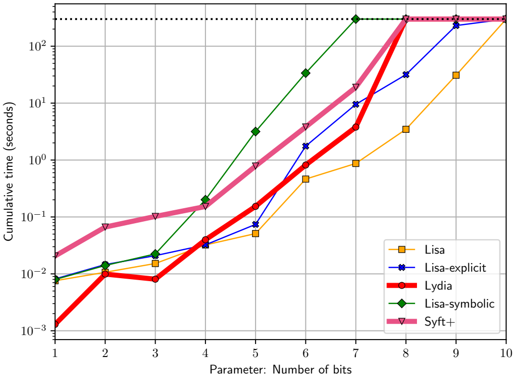
- Lydia/LydiaSyn
- MONA/Syft+
- Lisa ( only explicit, only symbolic, hybrid) (Bansal et al., 2020)

Datasets:

- Random conjunctions, 400 formulae (Zhu et al., 2017)
- Single counters, 20 formulae (Tabajara and Moshe Y Vardi, 2019)
- Double counters, 10 formulae (Tabajara and Moshe Y Vardi, 2019)
- Nim game, 24 formulae (Tabajara and Moshe Y Vardi, 2019)

# DFA Construction (single/double counters)



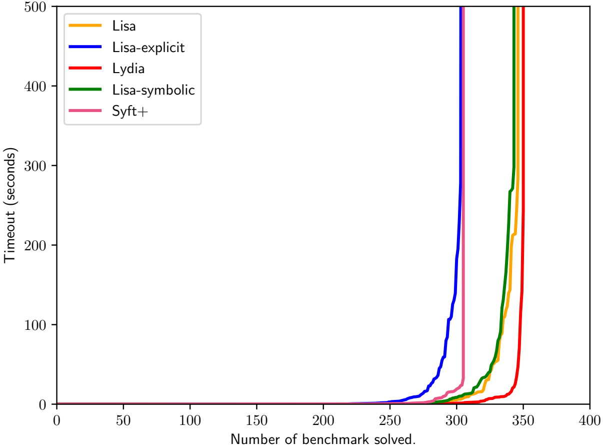



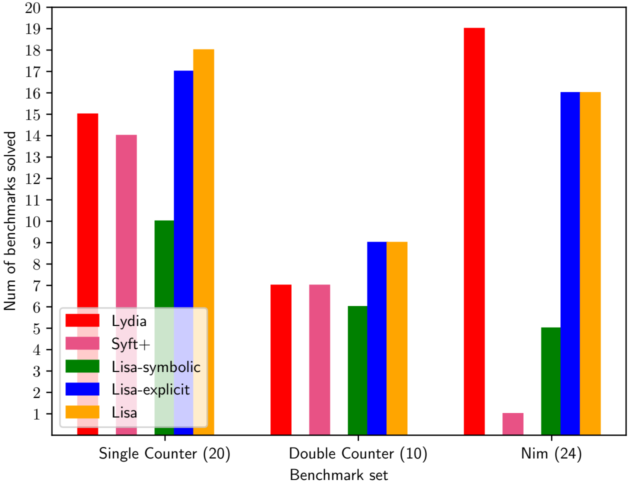
# DFA Construction (Nim game)

| Benchmark | | | | | |
|---|---|---|---|---|---|
| Name | Lydia | Mona-based | Lisa-explicit | Lisa-symbolic | Lisa |
| nim_1_1 | **0.01** | 0.15 | 0.07 | 0.07 | 0.07 |
| nim_1_2 | **0.02** | — | 0.15 | 0.16 | 0.16 |
| nim_1_3 | **0.05** | — | 0.07 | 1.43 | 0.06 |
| nim_1_4 | **0.09** | — | 0.14 | 267.23 | 0.13 |
| nim_1_5 | **0.17** | — | 0.27 | — | 0.25 |
| nim_1_6 | **0.30** | — | 0.63 | — | 0.54 |
| nim_1_7 | **0.54** | — | 1.20 | — | 1.02 |
| nim_1_8 | **0.82** | — | 1.87 | — | 1.83 |
| nim_2_1 | **0.05** | — | 0.14 | 1.49 | 0.10 |
| nim_2_2 | **0.20** | — | 0.84 | — | 0.81 |
| nim_2_3 | **1.47** | — | 4.95 | — | 4.95 |
| nim_2_4 | **7.00** | — | 26.07 | — | 24.33 |
| nim_2_5 | **34.86** | — | 125.56 | — | 108.86 |
| nim_2_6 | **114.87** | — | — | — | — |
| nim_2_7 | — | — | — | — | — |
| nim_3_1 | **0.40** | — | 3.15 | — | 2.67 |
| nim_3_2 | **9.93** | — | 84.34 | — | 78.31 |
| nim_3_3 | **142.16** | — | — | — | — |
| nim_3_4 | — | — | — | — | — |
| nim_4_1 | **8.97** | — | 110.10 | — | 109.79 |
| nim_4_2 | — | — | — | — | — |
| nim_5_1 | **243.62** | — | — | — | — |
| nim_5_2 | — | — | — | — | — |

Table 1: Running time (in seconds) for DFA construction on the Nim benchmark set. In bold the minimum running time for a given benchmark. — means time/memout. Timeout at 300 sec.

# DFA Construction, cactus plot




Num of benchmarks solved
20
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
Lydia
Syft+
Lisa-symbolic
Lisa-explicit
Lisa
Single Counter (20)
Double Counter (10)
Nim (24)
Benchmark set

# Conclusions and Future works

- Better than end-to-end MONA
  - Working directly with the right formalism gives better performances
- Fully compositional is (often) better
  - Lisa decomposes only in the outermost conjunction
- Heuristics are crucial for a scalable implementation
  - Agressive minimization (as in MONA)
  - Smallest products first (as in (Bansal et al., 2020))

**Future works:**

- Direct translations from $\text{LTL}_f$
- Direct translations for Past formulae ($\text{PLTL}_f$ and $\text{PLDL}_f$)
- Use a hybrid approach

- Bansal, Suguman et al. "Hybrid compositional reasoning for reactive synthesis from finite-horizon specifications". In: *AAAI*. 2020, pp. 9766–9774.
- Brafman, Ronen, Giuseppe De Giacomo, and Fabio Patrizi. "LTLf/LDLf Non-Markovian Rewards". In: (2018), pp. 1771–1778.
- De Giacomo, Giuseppe and Moshe Y. Vardi. "Linear Temporal Logic and Linear Dynamic Logic on Finite Traces". In: *IJCAI*. 2013, pp. 854–860.
- Henriksen, Jesper G. et al. "Mona: Monadic second-order logic in practice". In: 1995, pp. 89–110.
- Tabajara, Lucas Martinelli and Moshe Y Vardi. "Partitioning Techniques in LTLf Synthesis.". In: *IJCAI*. 2019, pp. 5599–5606.
- Yu, Fang et al. "Symbolic string verification: An automata-based approach". In: *SPIN*. 2008, pp. 306–324.
- Zhu, Shufang et al. "A symbolic approach to safety LTL synthesis". In: *HVC*. 2017.