Lydia: Compositional $\mathrm{LTL}_f/\mathrm{LDL}_f$ Synthesis

Marco Favorito

Bank of Italy
marco.favorito@bancaditalia.it
https://marcofavorito.me

Joint work with Giuseppe De Giacomo
(Published at ICAPS 2021)

The problem: LDL_f Synthesis (De Giacomo and M. Vardi, 2015)

- Alphabet partitioned in environment propositions and agent propositions
- Specification language (LDL_f), desirable (finite-trace) program executions
- **Output**: program procedure interacting with the environment, generated executions satisfy the LDL_f specification

Automata-theoretic solution:

- Compute the DFA of the LDL_f formula φ (double-exponential)
- Solve the DFA game (linear)

The problem: LDL_f Synthesis (De Giacomo and M. Vardi, 2015)

- Alphabet partitioned in environment propositions and agent propositions
- Specification language (LDL_f), desirable (finite-trace) program executions
- **Output**: program procedure interacting with the environment, generated executions satisfy the LDL_f specification

Automata-theoretic solution:

- Compute the DFA of the LDL_f formula φ (double-exponential)
- Solve the DFA game (linear)

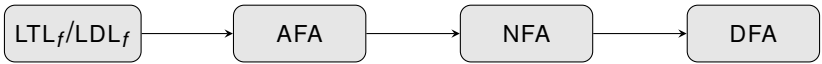
Focus: LDL_f-to-DFA construction

Given an LDL_f formula φ, compute a DFA $\mathcal{A}$ such that:

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

Related work

(De Giacomo and M. Vardi, 2013):



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

Our work:

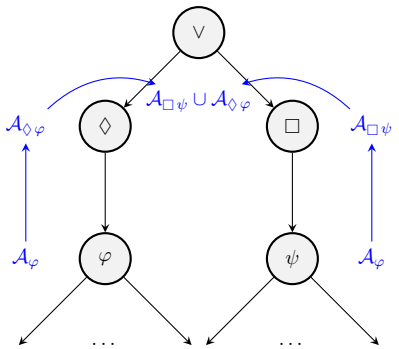
A new technique

- Fully compositional
  - Like (Bansal et al., 2020), but to the extreme
- Bottom-up approach
  - Against "top-down" approach of AFA-NFA-DFA
- non-elementary (instead of best theoretical bound of two-exp)
  - Yet, it works fairly well in practice
  - MONA too implements a non-elementary procedure!

How it works, in a nutshell

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

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



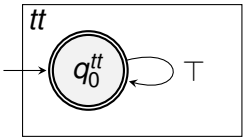
Example for $\langle \rho^* \rangle \varphi$ (test free)

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

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

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

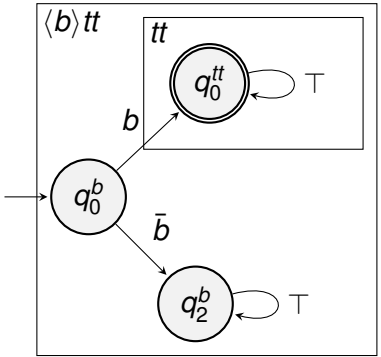
- Compute $\mathcal{A}_{tt}$



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

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

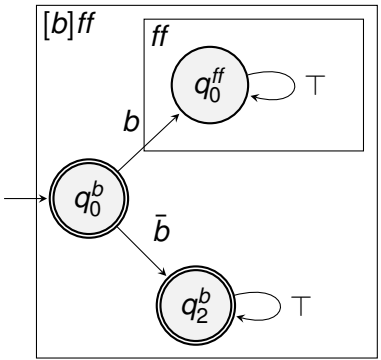
- Compute $\mathcal{A}_{\langle b \rangle tt}$



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

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

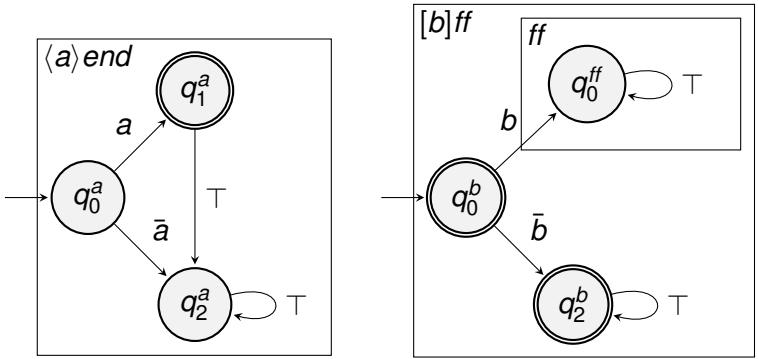
- Compute $\overline{\mathcal{A}_{\langle b \rangle tt}} = \mathcal{A}_{[b]ff}$



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

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

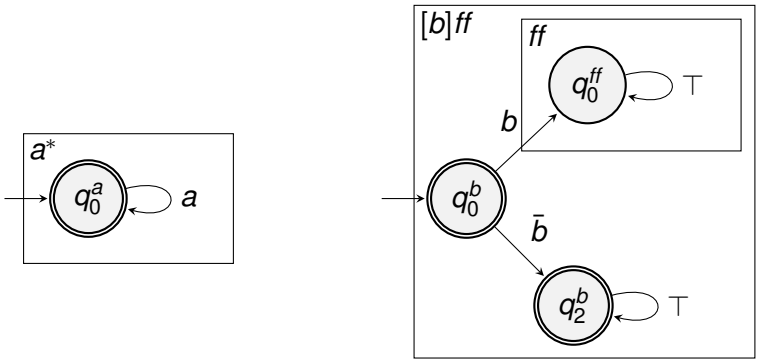
- Compute $\mathcal{A}_{\langle a \rangle end}$



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

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

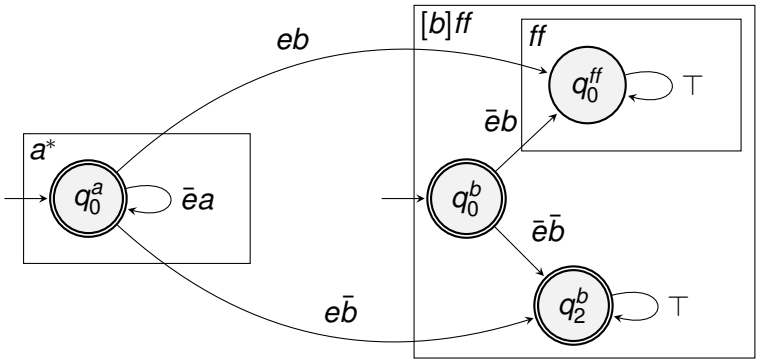
- Compute Kleene closure of $\mathcal{A}_{\langle a \rangle end}$, $\mathcal{A}_{a^*}$



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

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

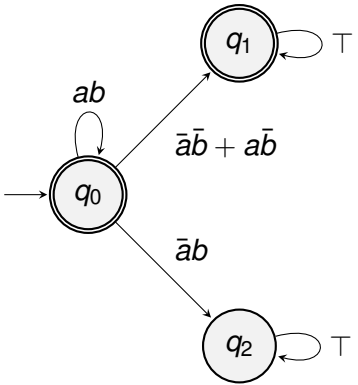
- Concatenate $\mathcal{A}_{a^*}$ and $\mathcal{A}_{[b]ff}$ (note: $\bar{e}a \wedge eb = \bot$)



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

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

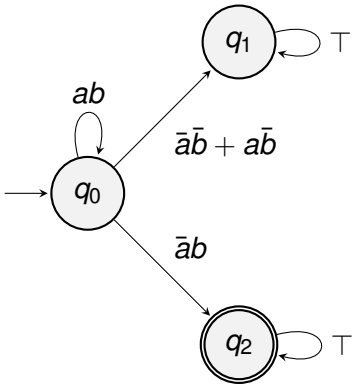
- Do EPROJECT($\mathcal{A}'_{\langle a^* \rangle [b] ff}$, i_e)



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

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

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



Implementation

Lydia

The technique has been implemented in a tool called **Lydia**[1]:

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

[1] https://github.com/whitemech/lydia

The MONA DFA library

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

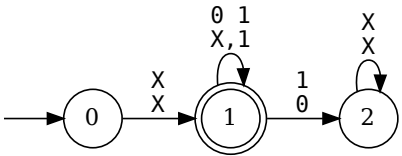
Lydia *only* uses the MONA DFA library. Features:

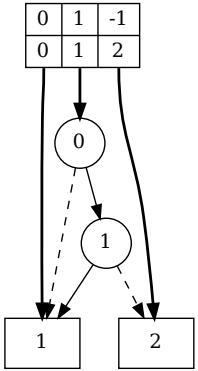
- manual construction of a DFA
- boolean operations between DFAs
- other operations: minimization, projections

The MONA DFA library (cont.)

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

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





Experiments

Benchmark

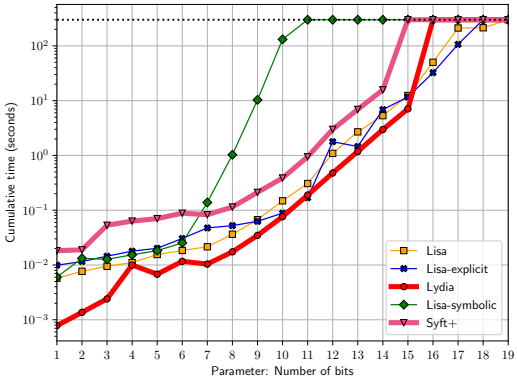
Tools:

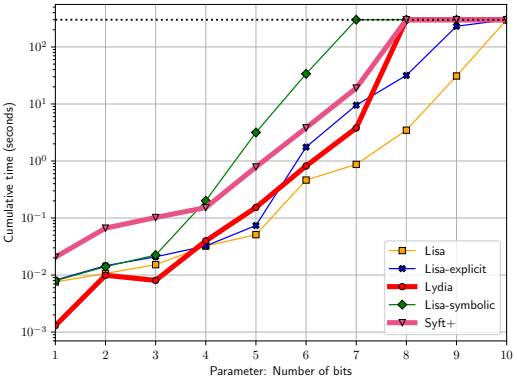
- Lydia/LydiaSyn
- MONA/Syft+
- Lisa (only explicit, only symbolic, hybrid) (Bansal et al., 2020)

Datasets:

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

DFA Construction (single/double counters)



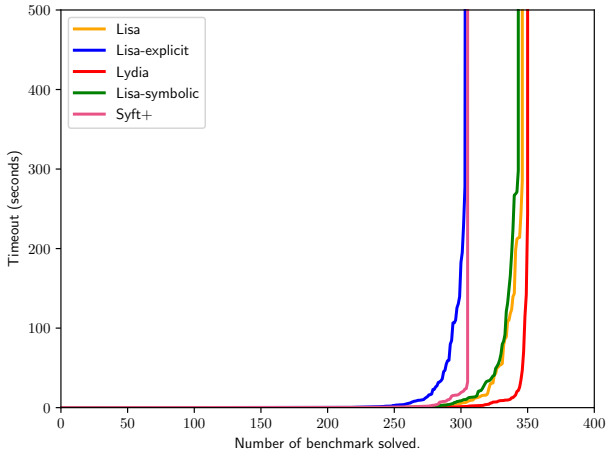


DFA Construction (Nim game)

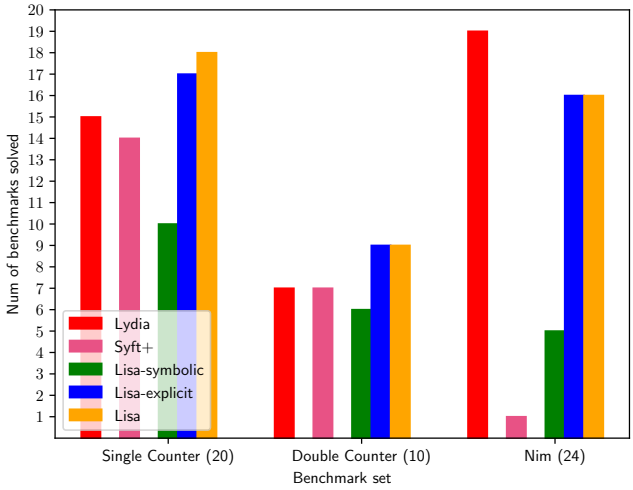
Benchmark					
Name	Lydia	Mona-based	Lisa-explicit	Lisa-symbolic	Lisa
nim_1_1	**0.01**	0.15	0.07	0.07	0.07
nim_1_2	**0.02**	—	0.15	0.16	0.16
nim_1_3	**0.05**	—	0.07	1.43	0.06
nim_1_4	**0.09**	—	0.14	267.23	0.13
nim_1_5	**0.17**	—	0.27	—	0.25
nim_1_6	**0.30**	—	0.63	—	0.54
nim_1_7	**0.54**	—	1.20	—	1.02
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



LTL_f Synthesis



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

Open source project: https://github.com/whitemech/lydia

References

- Bansal, Suguman et al. “Hybrid compositional reasoning for reactive synthesis from finite-horizon specifications”. In: *AAAI*. 2020, pp. 9766–9774.
- De Giacomo, Giuseppe and Moshe Y. Vardi. “Linear Temporal Logic and Linear Dynamic Logic on Finite Traces”. In: *IJCAI*. 2013, pp. 854–860.
- — .“Synthesis for LTL and LDL on Finite Traces”. In: *IJCAI*. 2015, pp. 1558–1564.
- 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.
- Zhu, Shufang et al. “A symbolic approach to safety LTL synthesis”. In: *HVC*. 2017.