Lydia: Compositional LTL_f/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)

Lydia: Compositional LTL_f/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)

Hi everyone! My name is Marco Favorito. The title of this presentation is ”Lydia: Compositional LTLf/LDLf Synthesis”. This is a work published at ICAPS in 2021, and it is a joint work with professor Giuseppe De Giacomo.

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)

We are interested in the problem of LDLf synthesis, that is, synthesis of linear dynamic logic formulas that are interpreted on finite traces. In this setting, we are given an alphabet of propositions, partitioned into environment and agent propositions, and a specification language to specify the desirable program executions. The solution to the problem is a procedure that controls the agent propositions in such a way that all the executions satisfy the LDLf specification.

The automata-theoretic solution is based on two steps: first, on the computation of the deterministic finite automaton of the LDL formula; and then, on solving the DFA game.

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:

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

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

This work focuses on the DFA computation step. This is the most expensive operation, since the size of the minimal automaton can be of double-exponentially larger than the size of the equivalent formula. The goal is to construct a DFA such that a trace satisfies the formula if and only if the trace is accepted by the automaton.

Related work

(De Giacomo and M. Vardi, 2013):

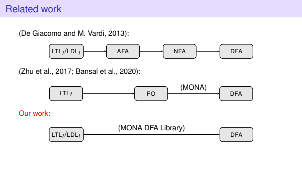
$\mathsf{LTL}_f/\mathsf{LDL}_f \longrightarrow \mathsf{AFA} \longrightarrow \mathsf{NFA} \longrightarrow \mathsf{DFA}$

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

$\mathsf{LTL}_f \longrightarrow \mathsf{FO} \xrightarrow{\text{(MONA)}} \mathsf{DFA}$

Our work:

$\mathsf{LTL}_f/\mathsf{LDL}_f \xrightarrow{\text{(MONA DFA Library)}} \mathsf{DFA}$



The classical approach for the DFA construction is to first convert the formula into an alternating finite automaton, and then determinize it.
Other works, like Zhu et al. 2017 and Bansal et al. 2020, exploit a first-order logic encoding of an LTLf formula, and then use the well-known MONA tool to produce a DFA.
In our work, we developed direct translation rules from LDLf formulas into DFAs. In our implementation, we used the MONA DFA library for automata operations.

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!

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!

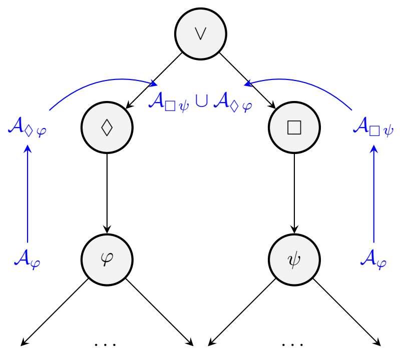
Our technique is fully compositional, since it is a bottom-up approach that starts from the smallest subformulas, and combines the partial results using automata-based operations.

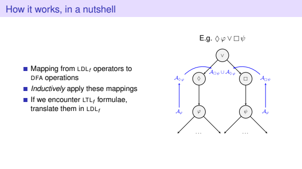
The time complexity of such technique is non-elementary, despite the optimal bound of double-exponential time. Yet, such technique works fairly well in practice; and note that the state-of-the-art MONA tool implements a non-elementary procedure.

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





This is how the technique works in a nutshell.
We devised a mapping from LDLf operators to DFA operators. The, we inductively apply this mapping throughout the syntax tree of the input formula. This technique works also for LTLf since we can linearly translate an LTLf formula into LDLf.

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

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

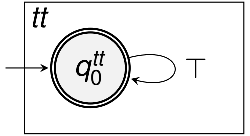
Example for $\langle \rho^* \rangle \varphi$ (test free)
Let $\varphi = [a^*]\langle b \rangle tt$

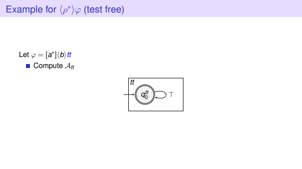
Here there is an example on how the technique works. Let the formula be box-a-star; diamond-b, true.

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

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

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



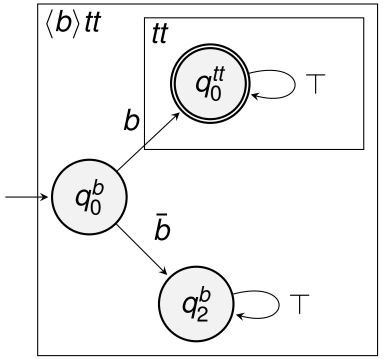


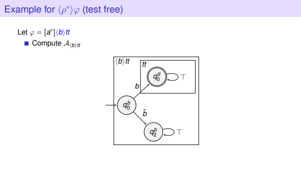
We start by computing the DFA of the elementary formula ”true”.

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

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

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



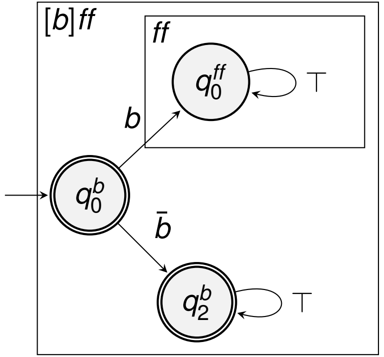


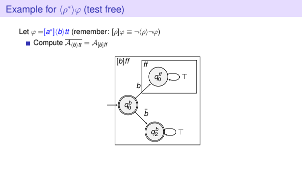
We then compute the DFA of the diamond formula with a propositional regular expression "b".

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



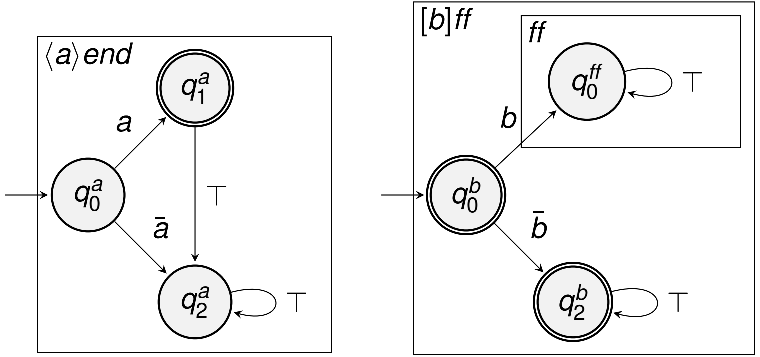


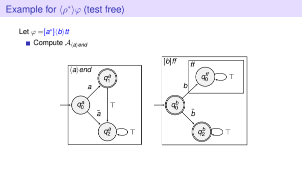
Next, we are going to translate the operator box-a-star. There is no to explain all the details, but the intuition is that

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

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

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



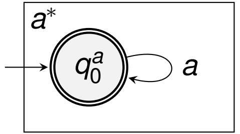


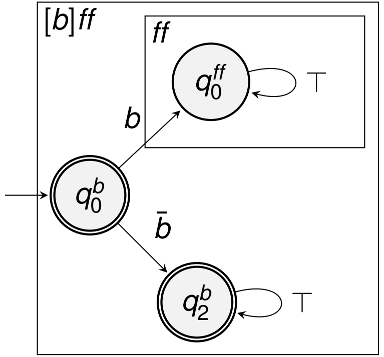
we compute the DFA of the kleene closure separately from the rest,

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

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

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





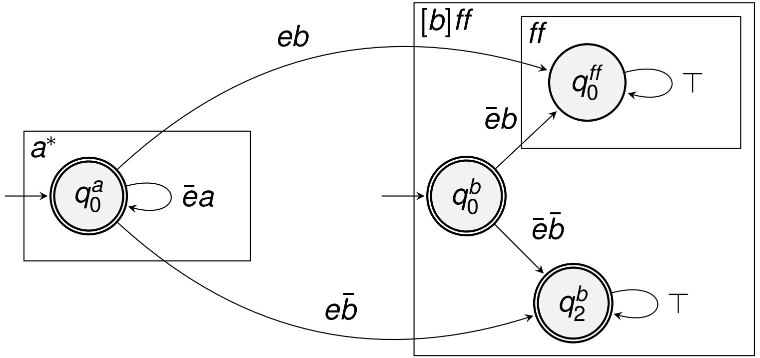
Example for $\langle \rho^* \rangle \varphi$ (test free)
Let $\varphi = [a^*]\langle b \rangle tt$
Compute Kleene closure of $\mathcal{A}_{\langle a \rangle end}$, $\mathcal{A}_{a^*}$

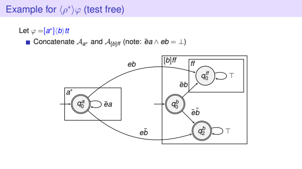
* pausa *

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: $\bar{e}a \wedge eb = \bot$)



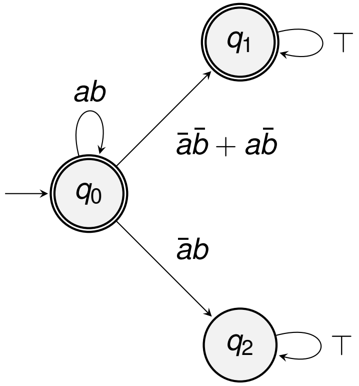


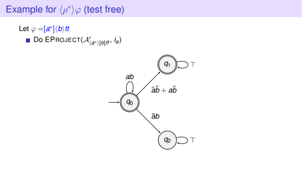
and then connect the two partial results with concatenating transitions. We use auxiliary bits to represent non-deterministic choices while staying in the same DFA formalism.

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)



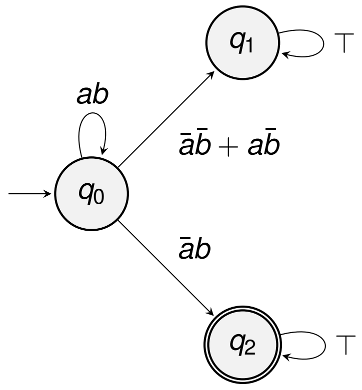


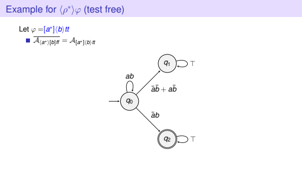
Once such transitions have been added, we project the auxiliary bits,

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

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

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





and then determinize the result.

Implementation

Next, we give a high-level overview of our implementation.

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 $\mathrm{LTL}_f/\mathrm{LDL}_f$ 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

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 $\mathrm{LTL}_f/\mathrm{LDL}_f$ 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 technique has been implemented in a tool called ”Lydia”.
It relies on MONA for DFA representation and operations over them.
It is integrated with Syft+ for solving the DFA game.
It depends on CUDD for certain subroutines.

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

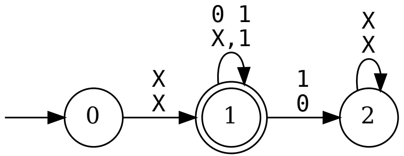
The MONA DFA library
MONA is a tool for translating Weak monadic Second-order theory of 1 Successor (WS1S) to DFAs.
Lydia *only* uses the MONA DFA library. Features:
- manual construction of a DFA
- boolean operations between DFAs
- other operations: minimization, projections

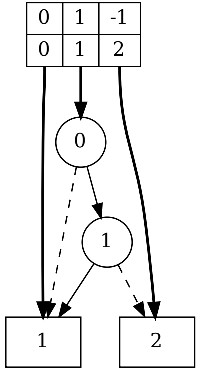
Lydia uses the MONA tool for manual construction of the DFA, boolean operations between DFAs, and other operations such as fast minimization and projection.

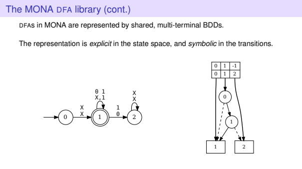
The MONA DFA library (cont.)

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

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







The DFAs in MONA are represented with shared multi-terminal binary decision diagrams.
The representation is explicit in the state space, and symbolic in the transitions.

Experiments

Next, we will show the experimental results on LTLf synthesis benchmark.

Benchmark

Tools:

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

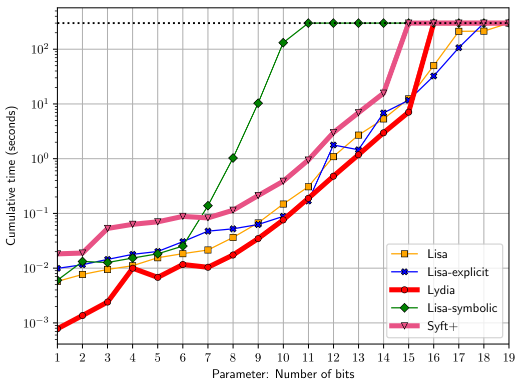
Datasets:

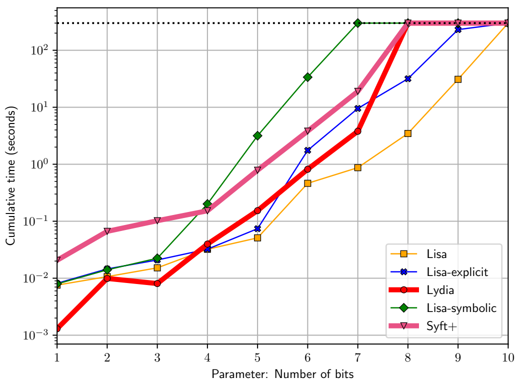
- Random conjunctions, 400 formulae (Zhu et al., 2017)
- Single counters, 20 formulae (Tabajara and M. Y. Vardi, 2019)
- Double counters, 10 formulae (Tabajara and M. Y. Vardi, 2019)
- Nim game, 24 formulae (Tabajara and M. Y. Vardi, 2019)

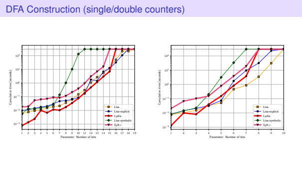
Benchmark
Tools:
Lydia/LydiaSyn
MONA/Syft+
Lisa (only explicit, only symbolic, hybrid) (Bansal et al., 2020)
Datasets:
Random conjunctions, 400 formulae (Zhu et al., 2017)
Single counters, 20 formulae (Tabajara and M. Y. Vardi, 2019)
Double counters, 10 formulae (Tabajara and M. Y. Vardi, 2019)
Nim game, 24 formulae (Tabajara and M. Y. Vardi, 2019)

We compared Lydia with MONA, and Lisa in its different modes: only explicit, only symbolic, and hybrid DFA construction.
We used several benchmarks known in the literature.

DFA Construction (single/double counters)







Regarding DFA construction, we see that Lydia is comparable with the other approaches, both for single counter and double counter set of formulas.

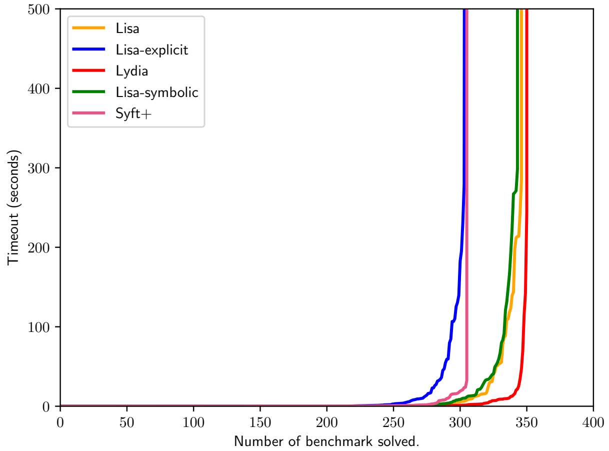
DFA Construction (Nim game)

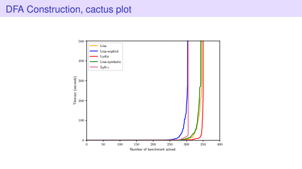
Benchmark					
Name	Lydia	Mona-based	Lisa-explicit	Lisa-symbolic	Lisa
nim_1_1	**0.01**	0.15	0.07	0.07	0.07
nim_1_2	**0.02**	—	0.15	0.16	0.16
nim_1_3	**0.05**	—	0.07	1.43	0.06
nim_1_4	**0.09**	—	0.14	267.23	0.13
nim_1_5	**0.17**	—	0.27	—	0.25
nim_1_6	**0.30**	—	0.63	—	0.54
nim_1_7	**0.54**	—	1.20	—	1.02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

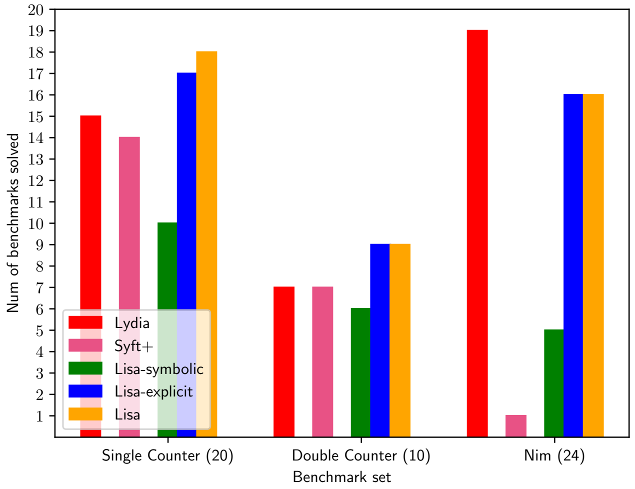
The same holds for the Nim benchmark, showing that our approach often performs better than the others.

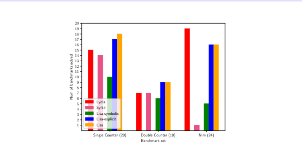
DFA Construction, cactus plot





This is the cactus plot showing the running times over the random formulas. This plot shows again that Lydia (the red line) is able to better handle this dataset.





Regarding the number of solved instances, once again we see that Lydia is comparable with its competitors on challenging benchmarks such as counters game and the Nim game.

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 LTL_f
- Direct translations for Past formulae (PLTL_f and PLDL_f)
- Use a hybrid approach

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

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 LTL_f
- Direct translations for Past formulae (PLTL_f and PLDL_f)
- Use a hybrid approach

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

To conclude, we observe that using direct translations are better than relying on first-order logic encoding. Moreover, a full compositional approach is often better since gives much more opportunity to minimize partial DFA results, and therefore tame the size of the DFA. The employed heuristics, e.g. aggressive minimization as in MONA, are crucial for scalability.

As a future work, we would like to devise direct translations for LTLf, direct translations for Pure-Past temporal logics, and explore the use of a hybrid approach as in Lisa.

The implementation is open source and can be found at this github.com/whitemech/lydia.

Note that you can use the software just for DFA construction, not necessarily also for synthesis.

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.

2023-03-26
Lydia: Compositional $\mathrm{LTL}_f/\mathrm{LDL}_f$ Synthesis
└References

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.