Alternating Nominal Automata with Name Allocation

23rd June 2025

Florian Frank, Daniel Hausmann, Stefan Milius,
Lutz Schröder and Henning Urbat

40th Annual ACM/IEEE Symposium on Logic in Computer Science; Singapore

Chair for Computer Science 8 (Theoretical Computer Science)
Friedrich-Alexander-Universität Erlangen-Nürnberg

T.CS

FAU Friedrich-Alexander-Universität
Faculty of Engineering

Motivation (Register Automata & Data Languages)

- Data languages are formal languages over an infinite alphabet.

$\mathbb{A}$: admissible user IDs for a server ($\rightsquigarrow$ *infinite set*)

- Data languages are formal languages over an infinite alphabet.

$\mathbb{A}$: admissible user IDs for a server ($\rightsquigarrow$ *infinite set*)

$$\mathcal{L} = \left\{ a_1 \cdots a_n \in \mathbb{A}^\star \ : \ \left(\begin{array}{c} a_1 = a_n \wedge \\ \forall 1 < i < n.\ a_1 \neq a_i \end{array} \right) \right\}$$

'first and last user coincide and differ from any other user'

T.CS

- Data languages are formal languages over an infinite alphabet.

  Standard model: Register Automata

$\mathbb{A}$: admissible user IDs for a server ($\rightsquigarrow$ *infinite set*)

$$\mathcal{L} = \left\{ a_1 \cdots a_n \in \mathbb{A}^{\star} \;:\; \left(\begin{array}{c} a_1 = a_n \wedge \\ \forall 1 < i < n.\, a_1 \neq a_i \end{array} \right) \right\}$$

'first and last user coincide and differ from any other user'

Motivation (Register Automata & Data Languages)

- Data languages are formal languages over an infinite alphabet.

  Standard model: Register Automata $\rightsquigarrow$ unfeasible for model checking (undecidable inclusion)

$\mathbb{A}$: admissible user IDs for a server ($\rightsquigarrow$ *infinite set*)

$$\mathcal{L} = \left\{ a_1 \cdots a_n \in \mathbb{A}^\star \ : \ \left(\begin{matrix} a_1 = a_n \wedge \\ \forall 1 < i < n.\ a_1 \neq a_i \end{matrix} \right) \right\}$$

'first and last user coincide and differ from any other user'

Motivation (Register Automata & Data Languages)

- Data languages are formal languages over an infinite alphabet.

  Standard model: Register Automata $\rightsquigarrow$ unfeasible for model checking (undecidable inclusion)

- To gain decidability, we must accept restrictions in their expressivity.

$\mathbb{A}$: admissible user IDs for a server ($\rightsquigarrow$ *infinite set*)

$$\mathcal{L} = \left\{ a_1 \cdots a_n \in \mathbb{A}^\star \; : \; \left(\begin{array}{c} a_1 = a_n \wedge \\ \forall 1 < i < n.\, a_1 \neq a_i \end{array} \right) \right\}$$

'first and last user coincide and differ from any other user'

Motivation (Register Automata & Data Languages)

T.CS

- Data languages are formal languages over an infinite alphabet.

  Standard model: Register Automata $\rightsquigarrow$ unfeasible for model checking (undecidable inclusion)

- To gain decidability, we must accept restrictions in their expressivity.

$\mathbb{A}$: admissible user IDs for a server ($\rightsquigarrow$ *infinite set*)

$$\mathcal{L} = \left\{ a_1 \cdots a_n \in \mathbb{A}^\star : \left(\begin{array}{c} a_1 = a_n \wedge \\ \forall 1 < i < n.\, a_1 \neq a_i \end{array} \right) \right\}$$

'first and last user coincide and differ from any other user'

Result — **Schröder, Kozen, Milius, Wißmann '17**

(Specific) **languages expressible by binding signatures** and their automata have decidable inclusion problems.

What are 'words with binders'?

- Data languages are formal languages over an infinite alphabet.

  Standard model: Register Automata ⇝ unfeasible for model checking (undecidable inclusion)

- To gain decidability, we must accept restrictions in their expressivity.

$\mathbb{A}$: admissible user IDs for a server (⇝ *infinite set*)

$$\mathcal{L} = \left\{ a_1 \cdots a_n \in \mathbb{A}^\star : \begin{pmatrix} a_1 = a_n \wedge \\ \forall 1 < i < n.\, a_1 \neq a_i \end{pmatrix} \right\}$$

'first and last user coincide and differ from any other user'

$\lambda a.(\lambda b.)^\star a$ (using shadowing)

Result — Schröder, Kozen, Milius, Wißmann '17

(Specific) **languages expressible by binding signatures** and their automata have decidable inclusion problems.

What are 'words with binders'?

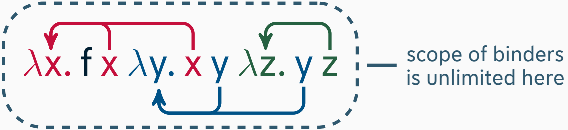
Motivation

- We consider data languages with explicit binders, which we see with λ-terms without parenthesis:

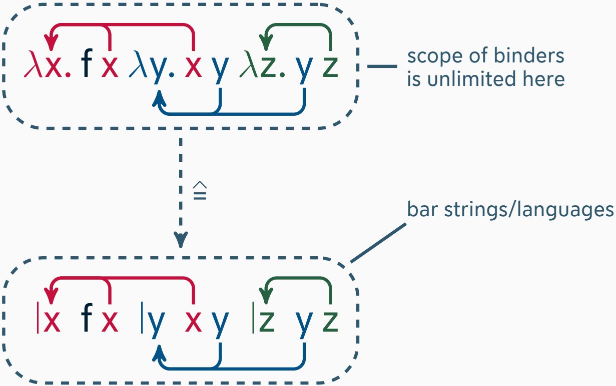
$$\lambda x.\ f\ x\ \lambda y.\ x\ y\ \lambda z.\ y\ z$$

scope of binders is unlimited here

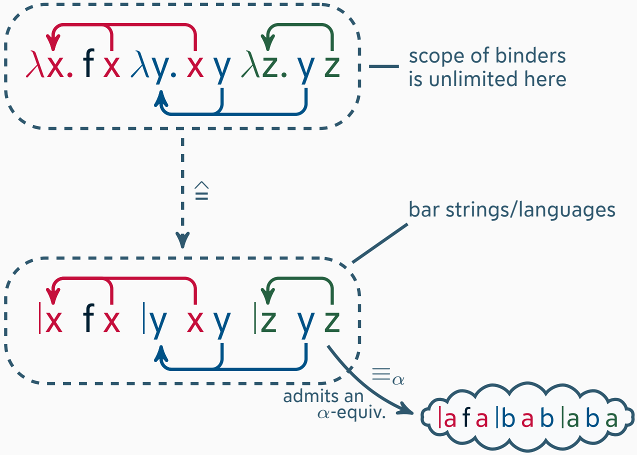
- We consider data languages with explicit binders, which we see with λ-terms without parenthesis:



- We consider data languages with explicit binders, which we see with λ-terms without parenthesis:

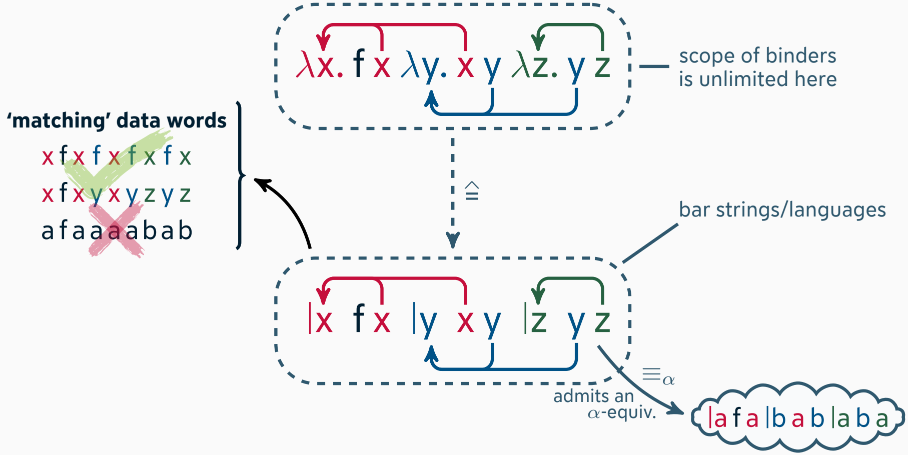


- We consider data languages with explicit binders, which we see with λ-terms without parenthesis:



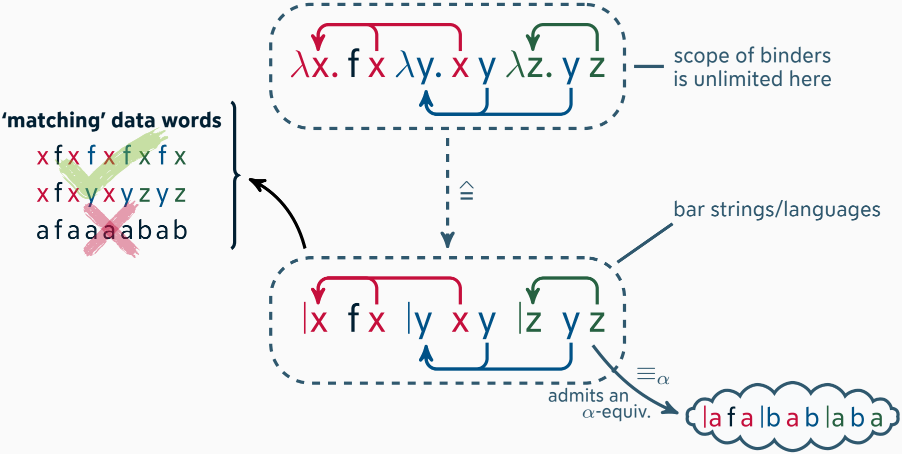
T.CS

- We consider data languages with explicit binders, which we see with λ-terms without parenthesis:



Motivation

- We consider data languages with explicit binders, which we see with λ-terms without parenthesis:



- 'Match' data words by taking any representative without bars.

Alternation For Bar Languages

Motivation

Remove de-alternation from model checking fixed-point logics over bar strings.

Motivation

Remove de-alternation from model checking fixed-point logics over bar strings.

Previous Approach: (Hausmann, Milius, Schröder '21)

linear-time fixed-point logic → Bar-μTL

Bar-μTL —doubly exponential blowup→ ERNNA

non-deterministic automata with $\top$-states → ERNNA

Alternation For Bar Languages

Motivation

Remove de-alternation from model checking fixed-point logics over bar strings.

» Previous Approach:

(Hausmann, Milius, Schröder '21)

linear-time fixed-point logic → Bar-μTL

Bar-μTL —— doubly exponential blowup ——→ ERNNA

non-deterministic automata with $\top$-states → ERNNA

Example of a Bar-μTL formula (over closed bar strings)

$$\varphi = \mu X.(\Diamond_{|a}(\Diamond_{|a}\top \vee \Diamond_a X \vee \varepsilon) \wedge \Diamond_{|a}\top)$$

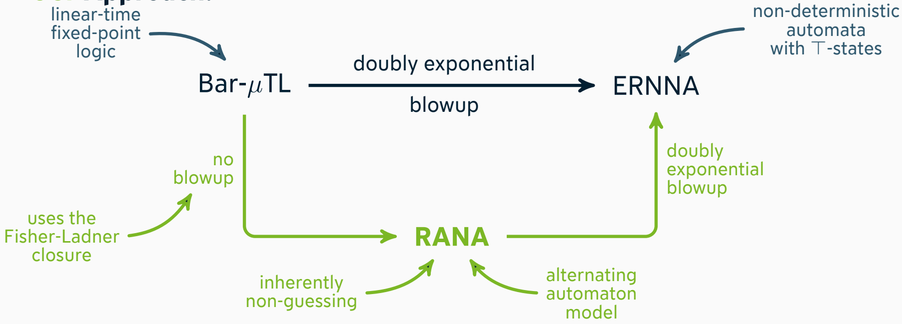
- $\Diamond_{|a}$, $\Diamond_a$: interpreted as 'consumes the letter' (modulo α-equiv.)
- φ: formulae are interpreted over finite bar strings ($w \models \varphi$)
- μX … X: 'recursion'
- ε: interpreted as 'string is empty'

» $|a|ba \models \varphi$ » $|aa|a \models \varphi$ » $|aa|ba \not\models \varphi$

Motivation

Remove de-alternation from model checking fixed-point logics over bar strings.

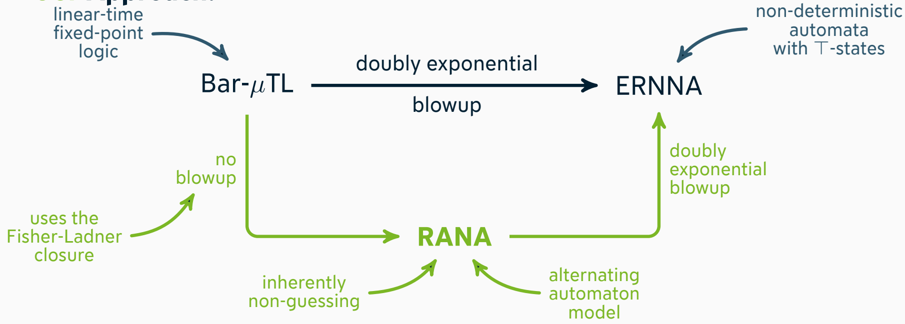
Our Approach:



Motivation

Remove de-alternation from model checking fixed-point logics over bar strings.

Our Approach:



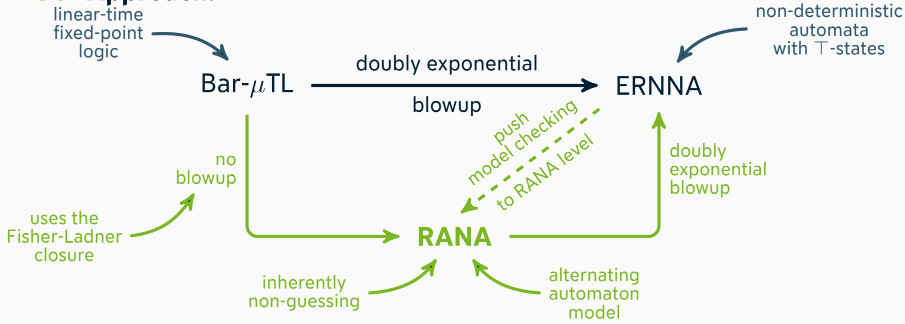
Theorem (*Expressive Equivalence*)

RANAs and Bar-μTL formulae are expressively equivalent.

Motivation

Remove de-alternation from model checking fixed-point logics over bar strings.

Our Approach:



Theorem (*Expressive Equivalence*)

RANAs and Bar-μTL formulae are expressively equivalent.

Alternation For Bar Languages

Motivation

Remove de-alternation from model checking fixed-point logics over bar strings.

» **Our Approach:**

linear-time fixed-point logic → Bar-μTL

uses the Fisher-Ladner closure → no blowup

alternating automaton model → **RANA**

Bar-μTL —(no blowup)→ **RANA**

Motivation

Remove de-alternation from model checking fixed-point logics over bar strings.

Our Approach:

linear-time fixed-point logic → Bar-μTL

Bar-μTL → (no blowup; uses the Fisher-Ladner closure) → **RANA** (alternating automaton model)

$$\varphi = \mu X.(\Diamond_{|a}(\Diamond_{|a}\top \vee \Diamond_{a} X \vee \varepsilon) \wedge \Diamond_{|a}\top)$$

Alternation For Bar Languages

Motivation

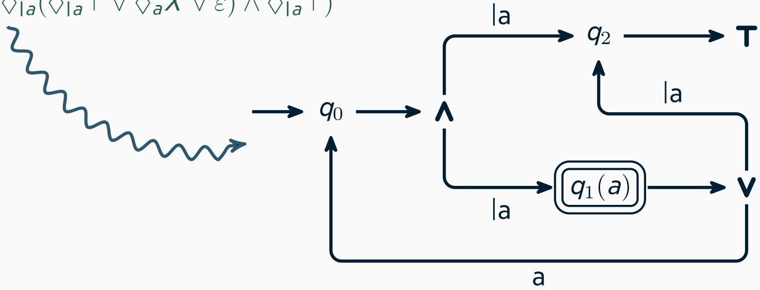
Remove de-alternation from model checking fixed-point logics over bar strings.

» Our Approach:

linear-time fixed-point logic → Bar-μTL —(no blowup)→ **RANA**

- no blowup: uses the Fisher-Ladner closure
- RANA: alternating automaton model

$$\varphi = \mu X.(\Diamond_{|a}(\Diamond_{|a}\top \vee \Diamond_a X \vee \varepsilon) \wedge \Diamond_{|a}\top)$$



Motivation

Remove de-alternation from model checking fixed-point logics over bar strings.

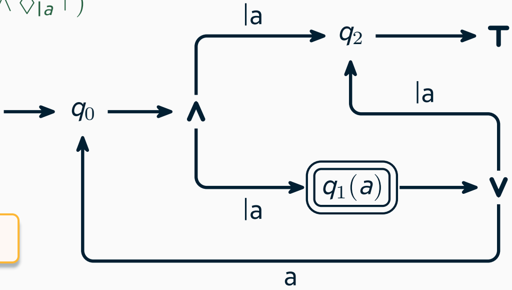
Our Approach:

linear-time fixed-point logic → Bar-μTL —no blowup→ **RANA**

uses the Fisher-Ladner closure

alternating automaton model

$$\varphi = \mu X.(\Diamond_{|a}(\Diamond_{|a}\top \vee \Diamond_a X \vee \varepsilon) \wedge \Diamond_{|a}\top)$$



Is $w = $ **|b b |c |d b** accepted?

Motivation

Remove de-alternation from model checking fixed-point logics over bar strings.

Our Approach:

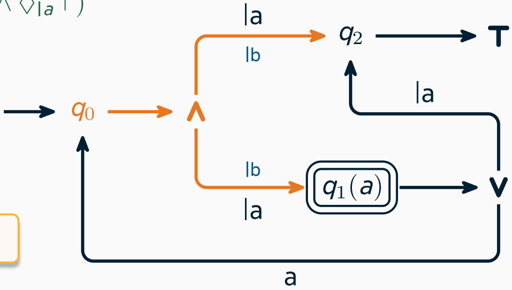
linear-time fixed-point logic → Bar-μTL

Bar-μTL —no blowup→ **RANA**

no blowup: uses the Fisher-Ladner closure

RANA: alternating automaton model

$$\varphi = \mu X.(\Diamond_{|a}(\Diamond_{|a}\top \vee \Diamond_{a} X \vee \varepsilon) \wedge \Diamond_{|a}\top)$$



Is $w = |\mathbf{b}\,\mathbf{b}\,|\mathbf{c}\,|\mathbf{d}\,\mathbf{b}$ accepted?

T.CS

Motivation

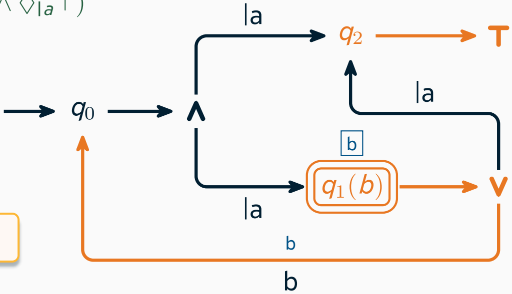
Remove de-alternation from model checking fixed-point logics over bar strings.

Our Approach:

linear-time fixed-point logic → Bar-μTL —— no blowup ——→ **RANA**

- no blowup: uses the Fisher-Ladner closure
- RANA: alternating automaton model

$$\varphi = \mu X.(\Diamond_{|a}(\Diamond_{|a}\top \vee \Diamond_{a}X \vee \varepsilon) \wedge \Diamond_{|a}\top)$$



Is $w = |\mathbf{b}\ \mathbf{b}\ |\mathbf{c}\ |\mathbf{d}\ \mathbf{b}$ accepted?

Alternation For Bar Languages

Motivation

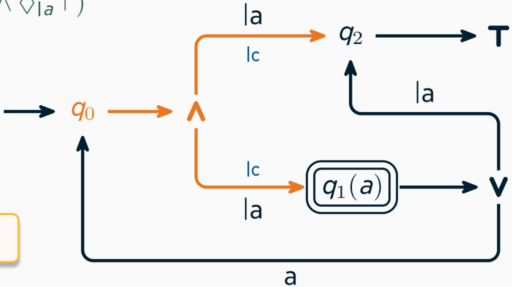
Remove de-alternation from model checking fixed-point logics over bar strings.

Our Approach:

linear-time fixed-point logic → Bar-μTL —(no blowup)→ **RANA**

- no blowup: uses the Fisher-Ladner closure
- RANA: alternating automaton model

$$\varphi = \mu X.(\Diamond_{|a}(\Diamond_{|a}\top \vee \Diamond_{a} X \vee \varepsilon) \wedge \Diamond_{|a}\top)$$



Is $w = $ |b b |c |d b accepted?

Alternation For Bar Languages

Motivation

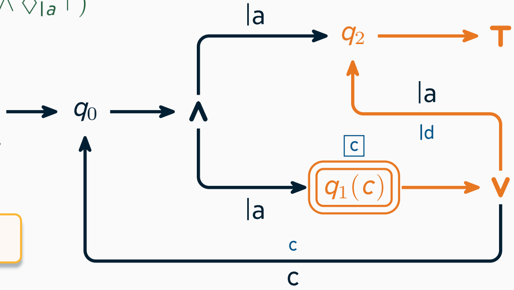
Remove de-alternation from model checking fixed-point logics over bar strings.

Our Approach:

Bar-μTL (linear-time fixed-point logic) → RANA (alternating automaton model)

no blowup — uses the Fisher-Ladner closure

$$\varphi = \mu X.(\Diamond_{|a}(\Diamond_{|a}\top \vee \Diamond_{a}X \vee \varepsilon) \wedge \Diamond_{|a}\top)$$



Is $w = |\mathbf{b}\,\mathbf{b}\,|\mathbf{c}\,|\mathbf{d}\,\mathbf{b}$ accepted?

Alternation For Bar Languages

Motivation

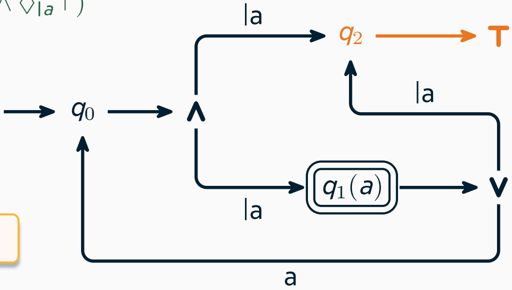
Remove de-alternation from model checking fixed-point logics over bar strings.

⋙ **Our Approach:**

linear-time fixed-point logic → Bar-μTL —no blowup→ **RANA**

- no blowup: uses the Fisher-Ladner closure
- RANA: alternating automaton model

$$\varphi = \mu X.(\Diamond_{|a}(\Diamond_{|a}\top \vee \Diamond_{a}X \vee \varepsilon) \wedge \Diamond_{|a}\top)$$



Is $w = $ |b b |c |d b accepted?

Motivation

Remove de-alternation from model checking fixed-point logics over bar strings.

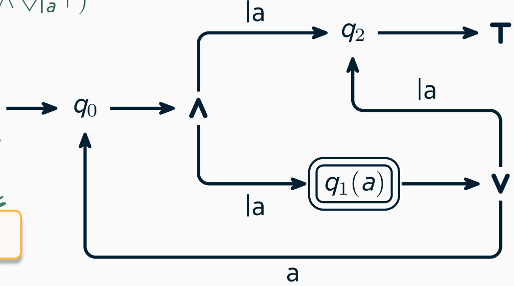
Our Approach:

linear-time fixed-point logic → Bar-μTL —(no blowup)→ **RANA**

uses the Fisher-Ladner closure

alternating automaton model

$\varphi = \mu X.(\Diamond_{|a}(\Diamond_{|a}\top \vee \Diamond_a X \vee \varepsilon) \wedge \Diamond_{|a}\top)$



Is $w = $ |**b** **b** |**c** |**d** **b** accepted?

De-Alternation

Idea

Restrict classical 'power-set construction' to sets of a fixed size.

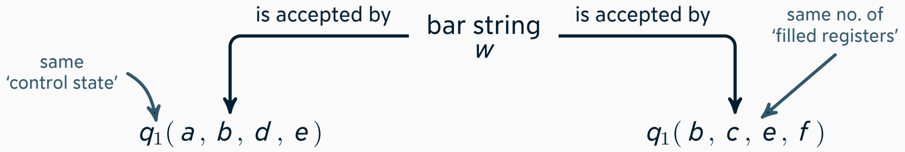
- How is this possible?

De-Alternation

Idea

Restrict classical ‘power-set construction’ to sets of a fixed size.

- How is this possible? (remove specific register values)



Idea

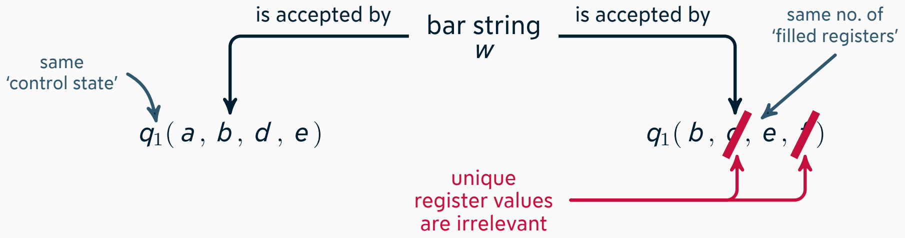
Restrict classical 'power-set construction' to sets of a fixed size.

» How is this possible? (remove specific register values)

Idea

Restrict classical ‘power-set construction’ to sets of a fixed size.

» How is this possible? (remove specific register values)

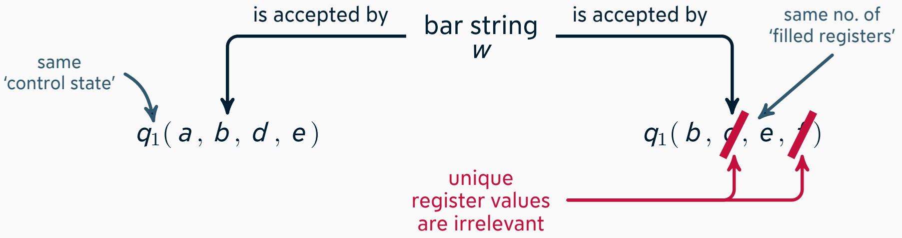


De-Alternation

Idea

Restrict classical 'power-set construction' to sets of a fixed size.

» How is this possible? (remove specific register values)



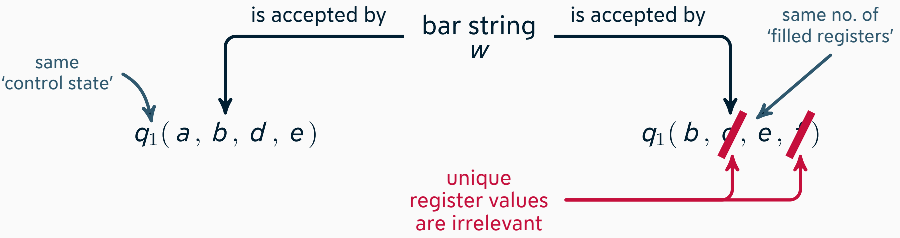
⇝ Iteratively, this results in at most singly exponentially many states.

De-Alternation

Idea

Restrict classical ‘power-set construction’ to sets of a fixed size.

» How is this possible? (remove specific register values)

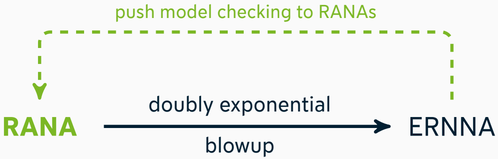


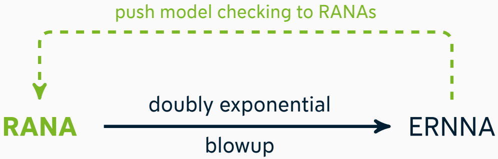
⇝ Iteratively, this results in at most singly exponentially many states.

Theorem (*De-Alternation*)

RANAs can be de-alternated to ERNNAs with a doubly exponential blowup.

Finite Representability

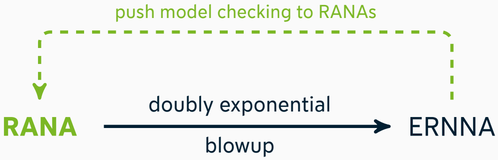




- With ERNNAs, model checking was done on the level of classical automata:

$$\text{ERNNA} \xleftrightarrow{\approx} \text{NFA with a } \top\text{-state}$$

Finite Representability



- With ERNNAs, model checking was done on the level of classical automata:

$$\text{ERNNA} \overset{\approx}{\longleftrightarrow} \text{NFA with a } \top\text{-state}$$

- To push model checking back to RANAs, we saw a similar ‘equivalence’:

Theorem (***Finitisation*** **)**

Every RANA has a non-emptiness equivalent classical AFA with exponentially many states.

in the no. of ‘registers’

Decidability Problems

RANA —exponential blowup (no. of registers)→ AFA

Decidability Problems

RANA —exponential blowup (no. of registers)→ AFA

- **Non-Emptiness Problem:** (decidable in ExpSpace)
  Solve via reduction to classical AFA problem.

Decidability Problems

RANA —exponential blowup (no. of registers)→ AFA

- **Non-Emptiness Problem:** (decidable in **ExpSpace**)
  Solve via reduction to classical AFA problem.
- **Universality Problem:** (decidable in **ExpSpace**)
  Check complement for (non-)emptiness.

Decidability Problems

RANA —exponential blowup (no. of registers)→ AFA

- **Non-Emptiness Problem:** (decidable in **ExpSpace**)
  Solve via reduction to classical AFA problem.
- **Universality Problem:** (decidable in **ExpSpace**)
  Check complement for (non-)emptiness.
- **Inclusion Problem:** (decidable in **ExpSpace**)
  Use well-known equivalence: $L_A \subseteq L_B$ iff $L_A \cap \mathbf{comp}(L_B) = \emptyset$.
  (complement of L_B → **comp**)

Decidability Problems

RANA —exponential blowup (no. of registers)→ AFA

- **Non-Emptiness Problem:** (decidable in ExpSpace)
  Solve via reduction to classical AFA problem.
- **Universality Problem:** (decidable in ExpSpace)
  Check complement for (non-)emptiness.
- **Inclusion Problem:** (decidable in ExpSpace)
  Use well-known equivalence: $L_A \subseteq L_B$ iff $L_A \cap \textbf{comp}(L_B) = \emptyset$. (complement of L_B)

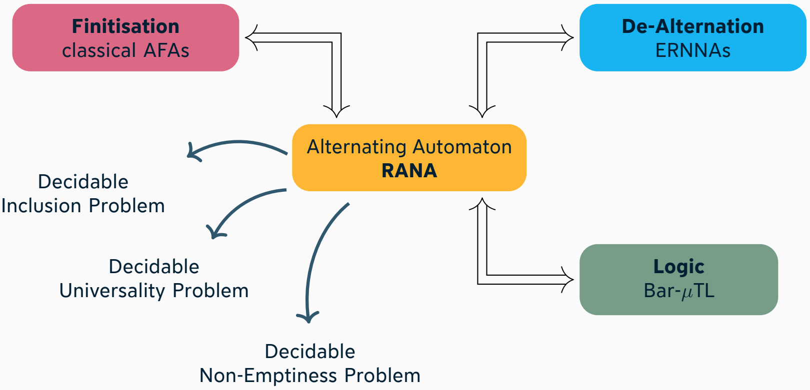
What about data languages?

For data languages, the inclusion problem is decidable in 2ExpSpace.

There is seemingly a need for de-alternation.

Conclusion & Future Work

- We looked at a variant of alternating automata for data languages with inherent name binding, and found many interesting properties:



- Some remaining problems:
  - Improve Local Freshness Complexity
  - Residuality/Learning RANAs
  - Extension to ω-Words

Questions?

Friedrich-Alexander-Universität
Faculty of Engineering

References

Hausmann, Daniel, Stefan Milius, Lutz Schröder. **'A Linear-Time Nominal μ-Calculus with Name Allocation'.** *46th International Symposium on Mathematical Foundations of Computer Science (MFCS 2021).* Ed. by Filippo Bonchi, Simon J. Puglisi. Vol. 202. LIPIcs. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, 58:1–58:18. ISBN: 978-3-95977-201-3. DOI: 10.4230/LIPIcs.MFCS.2021.58. URL: https://drops.dagstuhl.de/opus/volltexte/2021/14498.

Schröder, Lutz, Dexter Kozen, Stefan Milius, Thorsten Wißmann. **'Nominal Automata with Name Binding'.** *Proc. 20th International Conference on Foundations of Software Science and Computation Structures, (FOSSACS 2017).* Vol. 10203. Lect. Notes Comput. Sci. 2017, pp. 124–142.