Beyond $\mathcal{ALC}_{\mathsf{reg}}$: Exploring Non-Regular Extensions of PDL with DL Features

4th of September, DL Workshop 2023 & 22nd of September, JELIA 2023

Bartosz “Bart” Bednarczyk

With special thanks to Reijo Jaakkola, Witek Charatonik, and Sebastian Rudolph for all their support.

TU Dresden & University of Wrocław

Powered by BeamerikZ

Some historical results about $\mathcal{ALC}_{\mathsf{reg}}$ and beyond

Some historical results about $\mathcal{ALC}_{\mathsf{reg}}$ and beyond

$\mathcal{ALC}_{\mathsf{reg}} := \mathcal{ALC} + \exists\boldsymbol{\mathcal{L}}.\mathrm{C} + \forall\boldsymbol{\mathcal{L}}.\mathrm{C}$ for all languages $\boldsymbol{\mathcal{L}} \in \mathbb{REG}$.

Some historical results about $\mathcal{ALC}_{\mathsf{reg}}$ and beyond

$\mathcal{ALC}_{\mathsf{reg}} := \mathcal{ALC} + \exists\mathcal{L}.\mathrm{C} + \forall\mathcal{L}.\mathrm{C}$ for all languages $\mathcal{L} \in \mathbb{REG}$.

- Originally developed as a logic for program verification (a.k.a. Propositional Dynamic Logic)

Some historical results about $\mathcal{ALC}_{\mathsf{reg}}$ and beyond

$\mathcal{ALC}_{\mathsf{reg}} := \mathcal{ALC} + \exists\mathcal{L}.\mathrm{C} + \forall\mathcal{L}.\mathrm{C}$ for all languages $\mathcal{L} \in \mathbb{REG}$.

- Originally developed as a logic for program verification (a.k.a. Propositional Dynamic Logic)
- ExpTime-complete satisfiability (Pratt 1978).

Some historical results about $\mathcal{ALC}_{\mathsf{reg}}$ and beyond

$\mathcal{ALC}_{\mathsf{reg}} := \mathcal{ALC} + \exists\mathcal{L}.\mathrm{C} + \forall\mathcal{L}.\mathrm{C}$ for all languages $\mathcal{L} \in \mathbb{REG}$.

- Originally developed as a logic for program verification (a.k.a. Propositional Dynamic Logic)
- ExpTime-complete satisfiability (Pratt 1978).
- Robust under DL extensions (e.g. the $\mathcal{Z}$ family of DLs by Calvanese et al.)

Some historical results about $\mathcal{ALC}_{\text{reg}}$ and beyond

$\mathcal{ALC}_{\text{reg}} := \mathcal{ALC} + \exists\mathcal{L}.\mathrm{C} + \forall\mathcal{L}.\mathrm{C}$ for all languages $\mathcal{L} \in \mathbb{REG}$.

- Originally developed as a logic for program verification (a.k.a. Propositional Dynamic Logic)
- ExpTime-complete satisfiability (Pratt 1978).
- Robust under DL extensions (e.g. the $\mathcal{Z}$ family of DLs by Calvanese et al.)

Some historical results about $\mathcal{ALC}_{\mathsf{reg}}$ and beyond

$\mathcal{ALC}_{\mathsf{reg}} := \mathcal{ALC} + \exists\mathcal{L}.\mathrm{C} + \forall\mathcal{L}.\mathrm{C}$ for all languages $\mathcal{L} \in \mathbb{REG}$.

- Originally developed as a logic for program verification (a.k.a. Propositional Dynamic Logic)
- ExpTime-complete satisfiability (Pratt 1978).
- Robust under DL extensions (e.g. the $\mathcal{Z}$ family of DLs by Calvanese et al.)

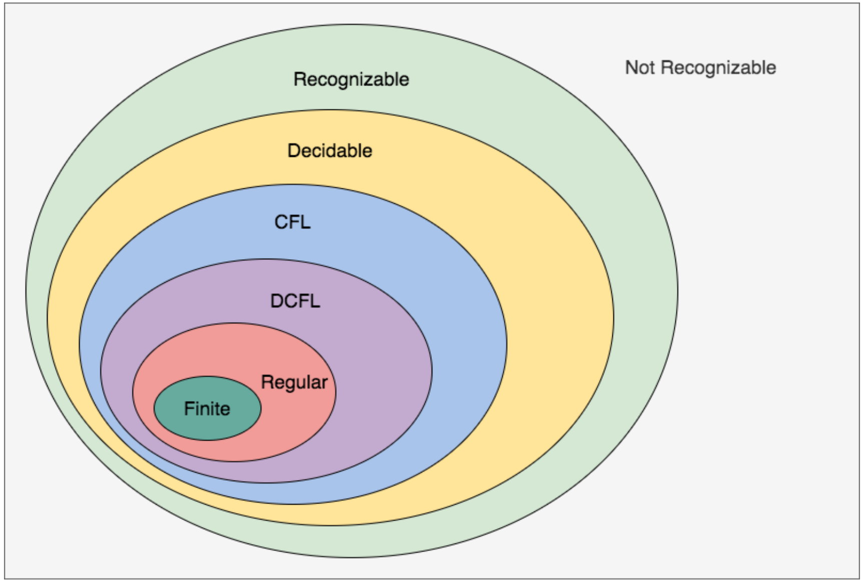
Can we go beyond regularity?

Some historical results about $\mathcal{ALC}_{\mathsf{reg}}$ and beyond

$$\mathcal{ALC}_{\mathsf{reg}} := \mathcal{ALC} + \exists\mathcal{L}.\mathrm{C} + \forall\mathcal{L}.\mathrm{C} \text{ for all languages } \mathcal{L} \in \mathbb{REG}.$$

- Originally developed as a logic for program verification (a.k.a. Propositional Dynamic Logic)
- ExpTime-complete satisfiability (Pratt 1978).
- Robust under DL extensions (e.g. the $\mathcal{Z}$ family of DLs by Calvanese et al.)

Can we go beyond regularity?

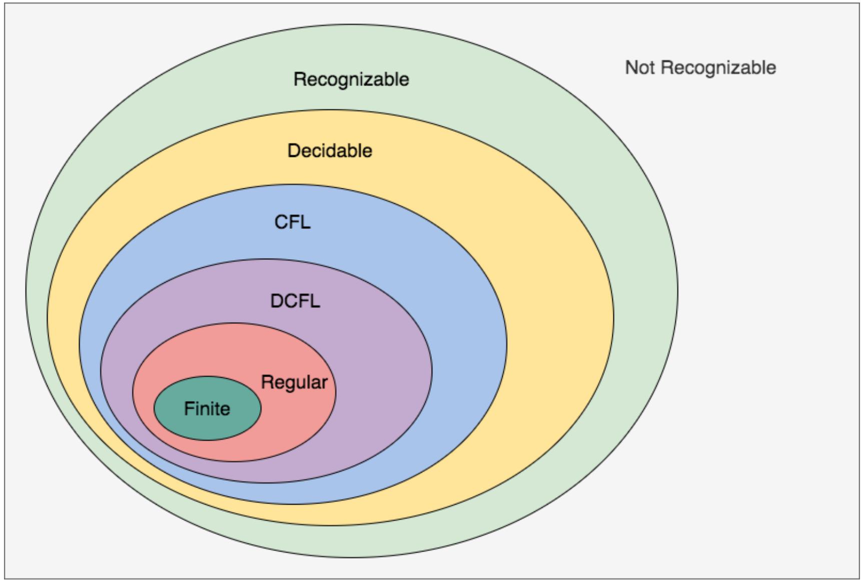


Some historical results about $\mathcal{ALC}_{\mathsf{reg}}$ and beyond

$$\mathcal{ALC}_{\mathsf{reg}} := \mathcal{ALC} + \exists\mathcal{L}.\mathrm{C} + \forall\mathcal{L}.\mathrm{C} \text{ for all languages } \mathcal{L} \in \mathbb{REG}.$$

- Originally developed as a logic for program verification (a.k.a. Propositional Dynamic Logic)
- ExpTime-complete satisfiability (Pratt 1978).
- Robust under DL extensions (e.g. the $\mathcal{Z}$ family of DLs by Calvanese et al.)

Can we go beyond regularity?



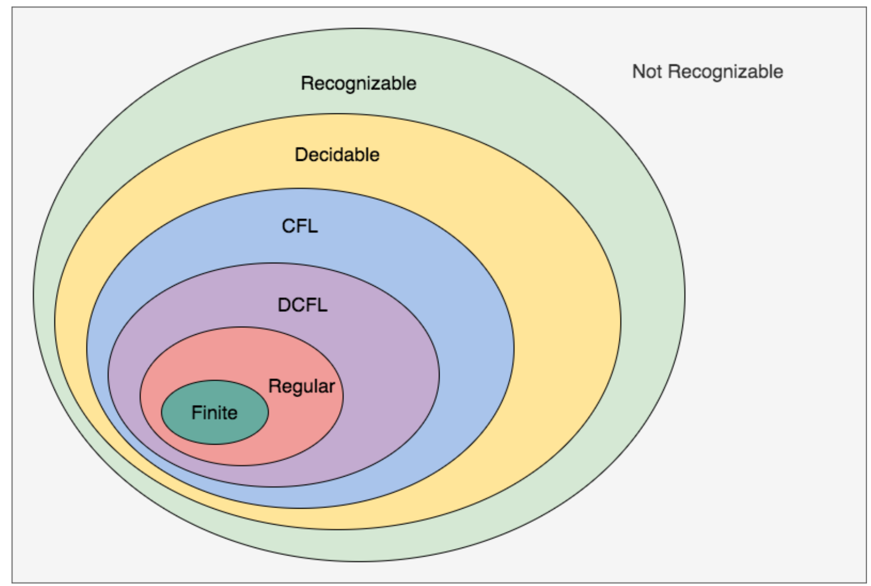
Some historical results about $\mathcal{ALC}_{\mathsf{reg}}$ and beyond

$\mathcal{ALC}_{\mathsf{reg}} := \mathcal{ALC} + \exists\mathcal{L}.\mathrm{C} + \forall\mathcal{L}.\mathrm{C}$ for all languages $\mathcal{L} \in \mathbb{REG}$.

- Originally developed as a logic for program verification (a.k.a. Propositional Dynamic Logic)
- ExpTime-complete satisfiability (Pratt 1978).
- Robust under DL extensions (e.g. the $\mathcal{Z}$ family of DLs by Calvanese et al.)

Can we go beyond regularity?

- $\mathbb{CFL}$ ('81)



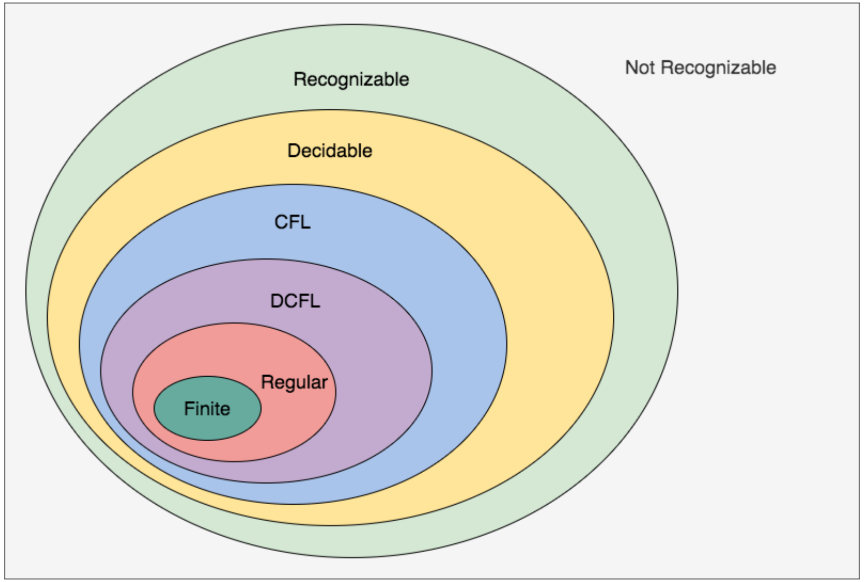
Some historical results about $\mathcal{ALC}_{\mathsf{reg}}$ and beyond

$$\mathcal{ALC}_{\mathsf{reg}} := \mathcal{ALC} + \exists\mathcal{L}.\mathrm{C} + \forall\mathcal{L}.\mathrm{C} \text{ for all languages } \mathcal{L} \in \mathbb{REG}.$$

- Originally developed as a logic for program verification (a.k.a. Propositional Dynamic Logic)
- ExpTime-complete satisfiability (Pratt 1978).
- Robust under DL extensions (e.g. the $\mathcal{Z}$ family of DLs by Calvanese et al.)

Can we go beyond regularity?

- $\mathbb{CFL}$ ('81)
- $\mathbb{REG} + r^{\#}sr^{\#}$



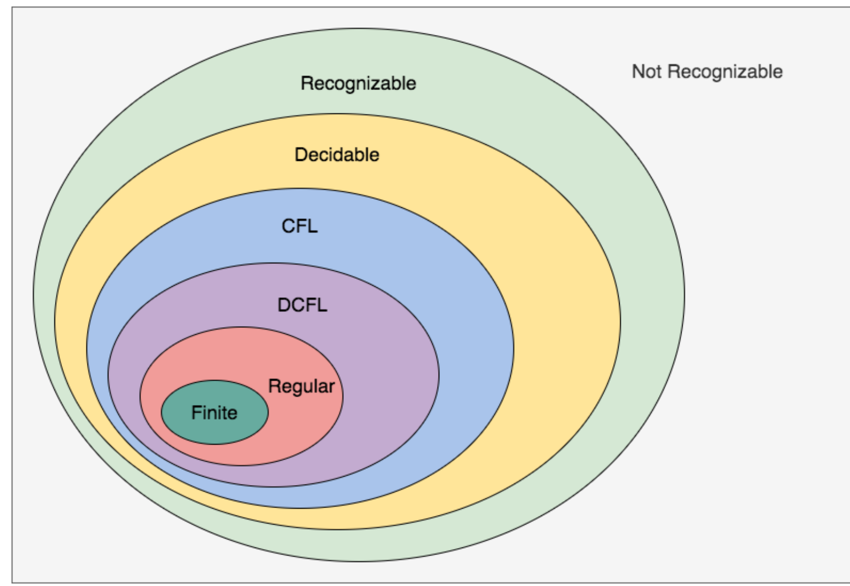
Some historical results about $\mathcal{ALC}_{\text{reg}}$ and beyond

$\mathcal{ALC}_{\text{reg}} := \mathcal{ALC} + \exists\mathcal{L}.\mathrm{C} + \forall\mathcal{L}.\mathrm{C}$ for all languages $\mathcal{L} \in \mathbb{REG}$.

- Originally developed as a logic for program verification (a.k.a. Propositional Dynamic Logic)
- ExpTime-complete satisfiability (Pratt 1978).
- Robust under DL extensions (e.g. the $\mathcal{Z}$ family of DLs by Calvanese et al.)

Can we go beyond regularity?

- $\mathbb{CFL}$ ('81)
- $\mathbb{REG} + r^{\#}sr^{\#}$
- $\mathbb{REG} + r^{\#}s^{\#} + s^{\#}r^{\#}$



Some historical results about $\mathcal{ALC}_{\text{reg}}$ and beyond

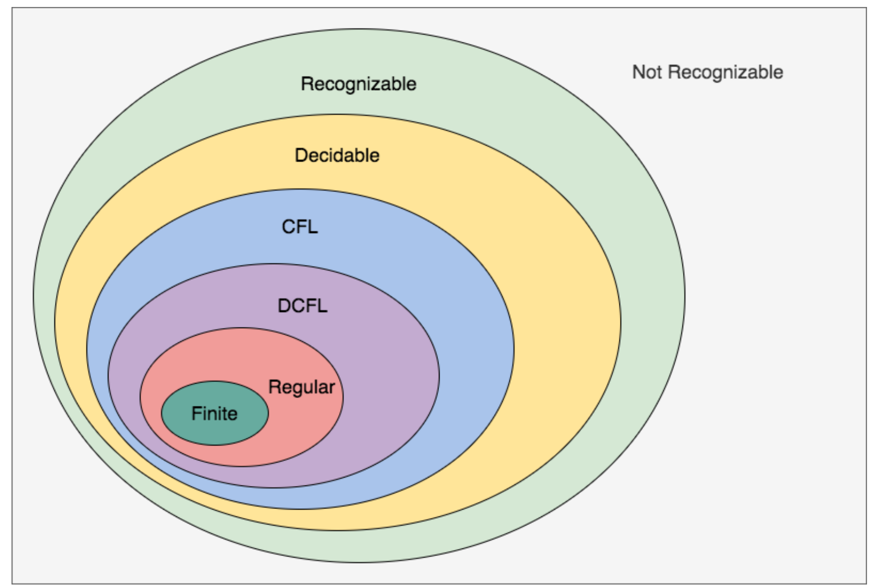
$$\mathcal{ALC}_{\text{reg}} := \mathcal{ALC} + \exists\mathcal{L}.\mathrm{C} + \forall\mathcal{L}.\mathrm{C} \text{ for all languages } \mathcal{L} \in \mathbb{REG}.$$

- Originally developed as a logic for program verification (a.k.a. Propositional Dynamic Logic)
- ExpTime-complete satisfiability (Pratt 1978).
- Robust under DL extensions (e.g. the $\mathcal{Z}$ family of DLs by Calvanese et al.)

Can we go beyond regularity?

- $\mathbb{CFL}$ ('81)
- $\mathbb{REG} + r^{\#}sr^{\#}$
- $\mathbb{REG} + r^{\#}s^{\#} + s^{\#}r^{\#}$

- $\mathbb{REG} + r^{\#}s^{\#}$



Some historical results about $\mathcal{ALC}_{\text{reg}}$ and beyond

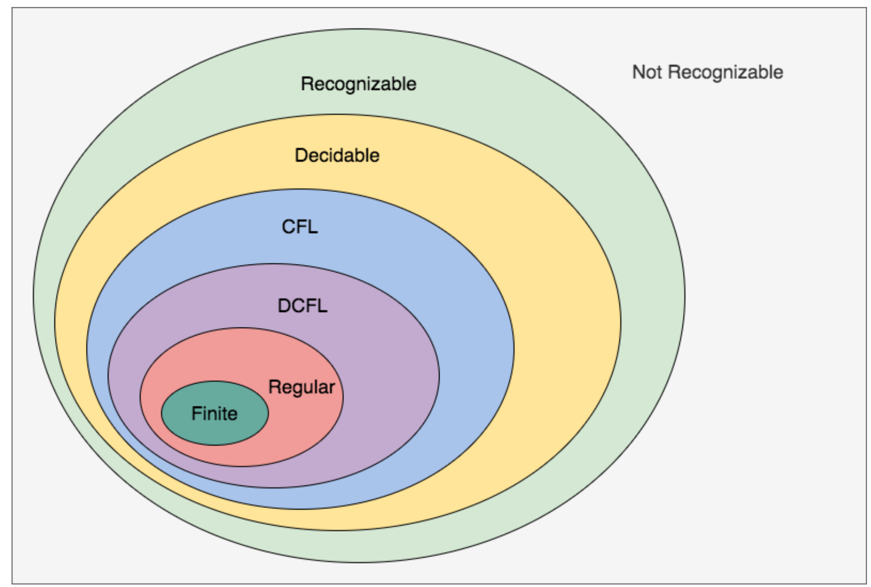
$$\mathcal{ALC}_{\text{reg}} := \mathcal{ALC} + \exists\mathscr{L}.\mathrm{C} + \forall\mathscr{L}.\mathrm{C} \text{ for all languages } \mathscr{L} \in \mathbb{REG}.$$

- Originally developed as a logic for program verification (a.k.a. Propositional Dynamic Logic)
- ExpTime-complete satisfiability (Pratt 1978).
- Robust under DL extensions (e.g. the $\mathcal{Z}$ family of DLs by Calvanese et al.)

Can we go beyond regularity?

- $\mathbb{CFL}$ ('81)
- $\mathbb{REG} + r^{\#}sr^{\#}$
- $\mathbb{REG} + r^{\#}s^{\#} + s^{\#}r^{\#}$

- $\mathbb{REG} + r^{\#}s^{\#}$
- $\mathbb{REG}$ + (semi) simple minded



Some historical results about $\mathcal{ALC}_{\mathsf{reg}}$ and beyond

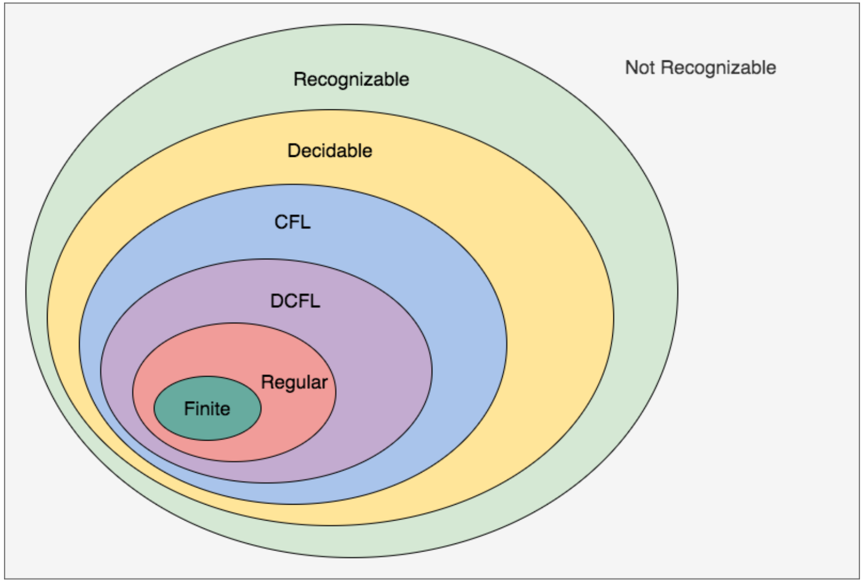
$$\mathcal{ALC}_{\mathsf{reg}} := \mathcal{ALC} + \exists\mathcal{L}.\mathrm{C} + \forall\mathcal{L}.\mathrm{C} \text{ for all languages } \mathcal{L} \in \mathbb{REG}.$$

- Originally developed as a logic for program verification (a.k.a. Propositional Dynamic Logic)
- ExpTime-complete satisfiability (Pratt 1978).
- Robust under DL extensions (e.g. the $\mathcal{Z}$ family of DLs by Calvanese et al.)

Can we go beyond regularity?

- $\mathbb{CFL}$ ('81)
- $\mathbb{REG} + r^{\#}sr^{\#}$
- $\mathbb{REG} + r^{\#}s^{\#} + s^{\#}r^{\#}$

- $\mathbb{REG} + r^{\#}s^{\#}$
- $\mathbb{REG}$ + (semi) simple minded
- More...



Some historical results about $\mathcal{ALC}_{\mathsf{reg}}$ and beyond

$\mathcal{ALC}_{\mathsf{reg}} := \mathcal{ALC} + \exists\mathcal{L}.\mathrm{C} + \forall\mathcal{L}.\mathrm{C}$ for all languages $\mathcal{L} \in \mathbb{REG}$.

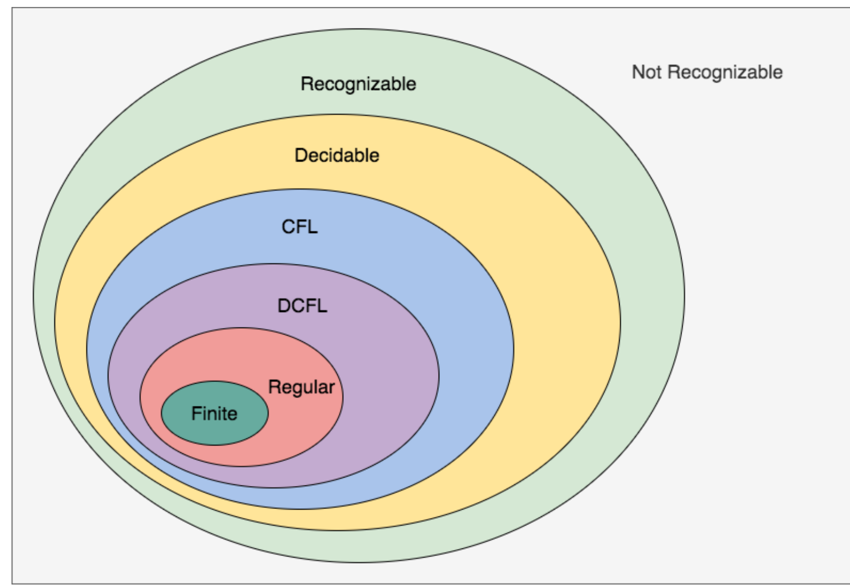
- Originally developed as a logic for program verification (a.k.a. Propositional Dynamic Logic)
- ExpTime-complete satisfiability (Pratt 1978).
- Robust under DL extensions (e.g. the $\mathcal{Z}$ family of DLs by Calvanese et al.)

Can we go beyond regularity?

- $\mathbb{CFL}$ ('81)
- $\mathbb{REG} + r^{\#}sr^{\#}$
- $\mathbb{REG} + r^{\#}s^{\#} + s^{\#}r^{\#}$

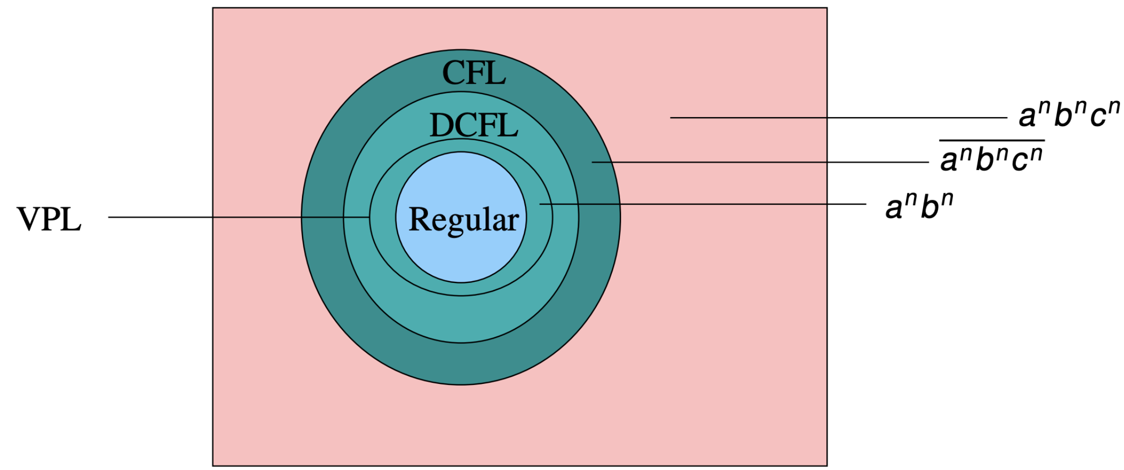
- $\mathbb{REG} + r^{\#}s^{\#}$
- $\mathbb{REG}$ + (semi) simple minded
- More...

$\mathcal{ALC}$+Visibly Pushdown Languages is decidable.

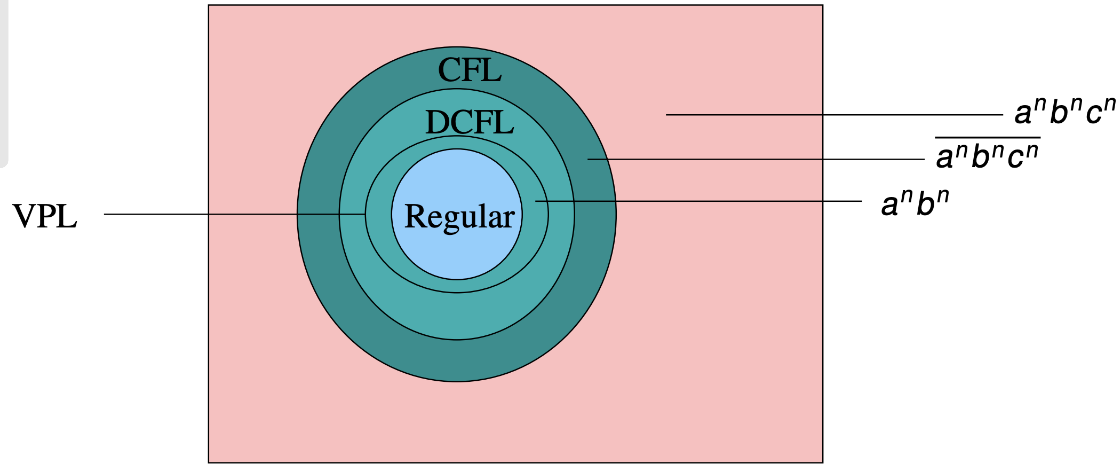


The success of Visibly Pushdown Languages (VPLs)

The success of Visibly Pushdown Languages (VPLs)

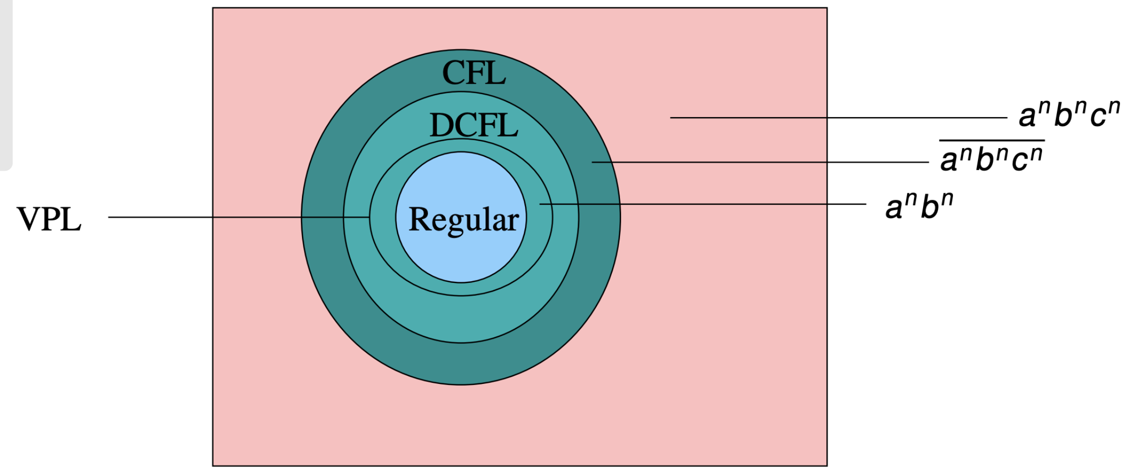


The success of Visibly Pushdown Languages (VPLs)



The success of Visibly Pushdown Languages (VPLs)

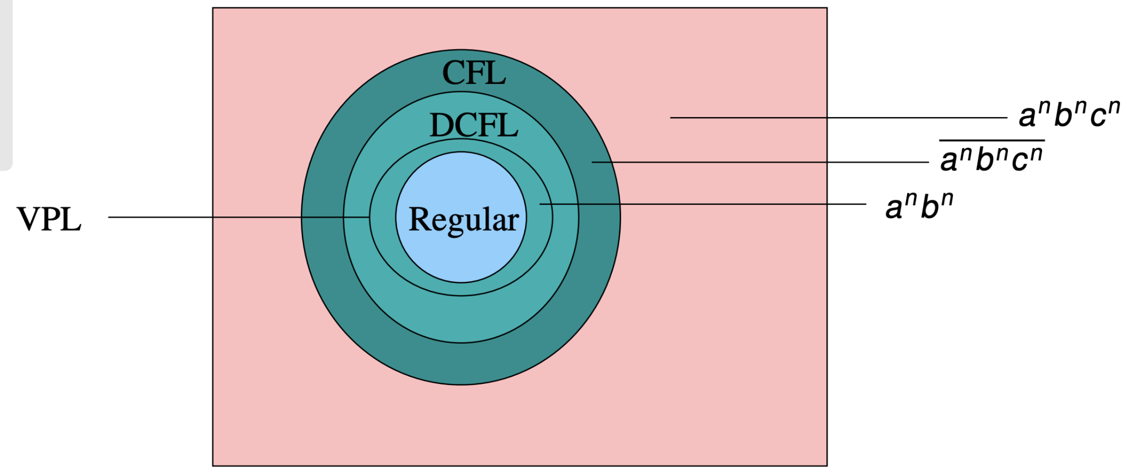
$\mathbb{VPL}$ = Finite Automata + Input-Driven Stack



The success of Visibly Pushdown Languages (VPLs)

$\mathbb{VPL}$ = Finite Automata + Input-Driven Stack

$\Sigma = \Sigma_c \cup \Sigma_i \cup \Sigma_r$ (call + internal + returns)

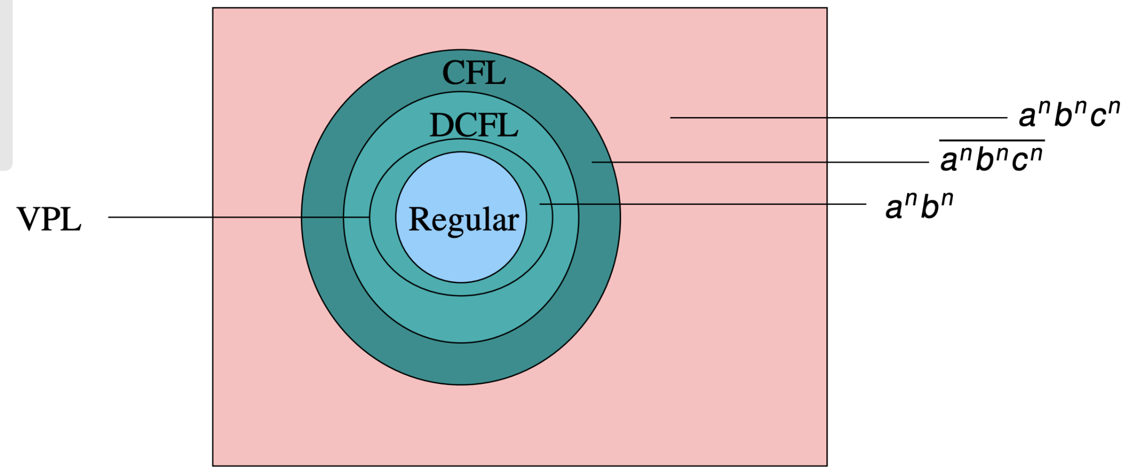


The success of Visibly Pushdown Languages (VPLs)

$\mathbb{VPL}$ = Finite Automata + Input-Driven Stack

$\Sigma = \Sigma_c \cup \Sigma_i \cup \Sigma_r$ (call + internal + returns)

Push (pop) only after reading a call (return).



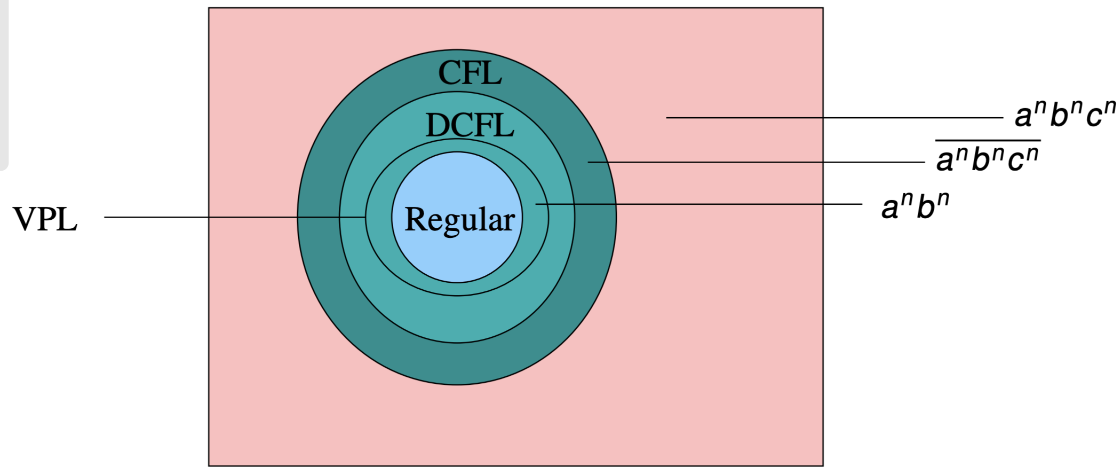
The success of Visibly Pushdown Languages (VPLs)

$\mathbb{VPL}$ = Finite Automata + Input-Driven Stack

$\Sigma = \Sigma_c \cup \Sigma_i \cup \Sigma_r$ (call + internal + returns)

Push (pop) only after reading a call (return).

- Ex1: Dyck languages



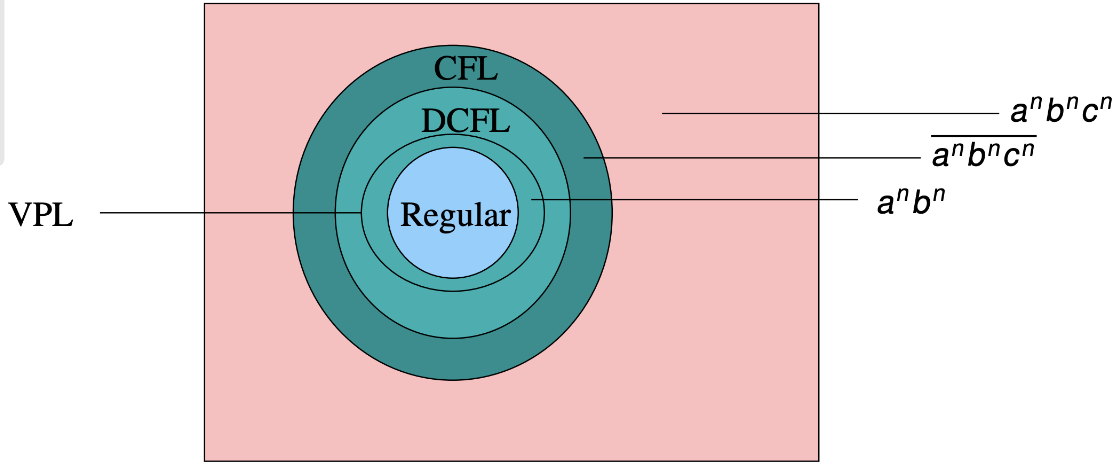
The success of Visibly Pushdown Languages (VPLs)

$\mathbb{VPL}$ = Finite Automata + Input-Driven Stack

$\Sigma = \Sigma_c \cup \Sigma_i \cup \Sigma_r$ (call + internal + returns)

Push (pop) only after reading a call (return).

- Ex1: Dyck languages
- Ex2: $c^{\#}r^{\#}$ but not $r^{\#}c^{\#}$ (for $c \in \Sigma_c, r \in \Sigma_r$)



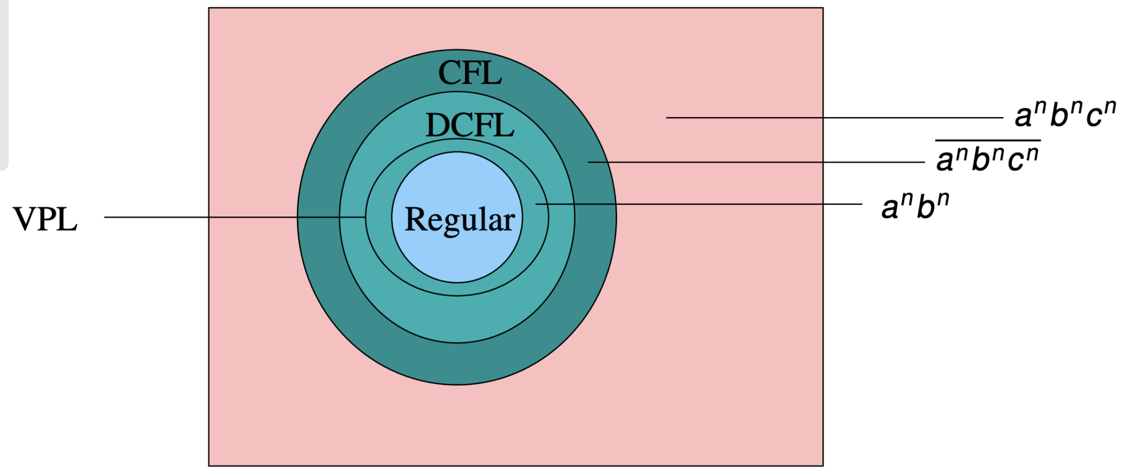
The success of Visibly Pushdown Languages (VPLs)

$\mathbb{VPL}$ = Finite Automata + Input-Driven Stack

$\Sigma = \Sigma_c \cup \Sigma_i \cup \Sigma_r$ (call + internal + returns)

Push (pop) only after reading a call (return).

- Ex1: Dyck languages
- Ex2: $c^{\#} r^{\#}$ but not $r^{\#} c^{\#}$ (for $c \in \Sigma_c, r \in \Sigma_r$)
- Ex3: Every regular language is in $\mathbb{VPL}$



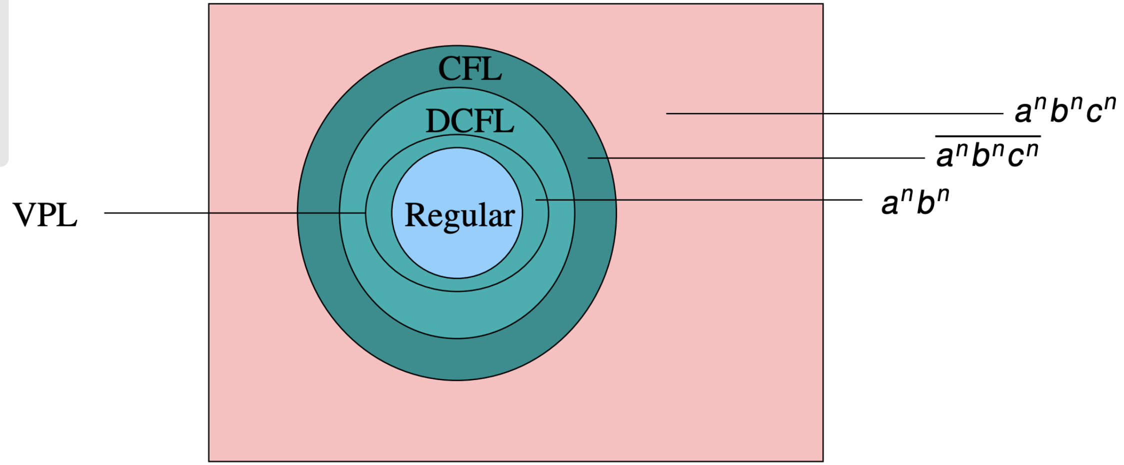
The success of Visibly Pushdown Languages (VPLs)

$\mathbb{VPL}$ = Finite Automata + Input-Driven Stack

$\Sigma = \Sigma_c \cup \Sigma_i \cup \Sigma_r$ (call + internal + returns)

Push (pop) only after reading a call (return).

- Ex1: Dyck languages
- Ex2: $c^{\#}r^{\#}$ but not $r^{\#}c^{\#}$ (for $c \in \Sigma_c, r \in \Sigma_r$)
- Ex3: Every regular language is in $\mathbb{VPL}$



Why do we care?

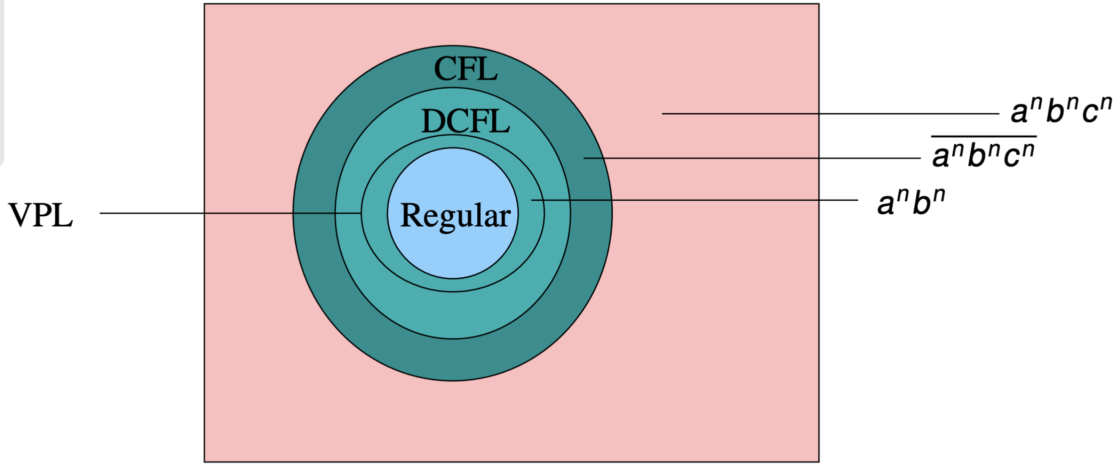
The success of Visibly Pushdown Languages (VPLs)

$\mathbb{VPL}$ = Finite Automata + Input-Driven Stack

$\Sigma = \Sigma_c \cup \Sigma_i \cup \Sigma_r$ (call + internal + returns)

Push (pop) only after reading a call (return).

- Ex1: Dyck languages
- Ex2: $c^{\#}r^{\#}$ but not $r^{\#}c^{\#}$ (for $c \in \Sigma_c, r \in \Sigma_r$)
- Ex3: Every regular language is in $\mathbb{VPL}$



Why do we care?

	Decision problems for automata		
	Emptiness	Universality/Equivalence	Inclusion
NFA	NLOGSPACE	PSPACE	PSPACE
PDA	PTIME	Undecidable	Undecidable
DPDA	PTIME	Decidable	Undecidable
VPA	PTIME	EXPTIME	EXPTIME

	Closure under				
	Union	Intersection	Complement	Concat.	Kleene-$*$
Regular	Yes	Yes	Yes	Yes	Yes
CFL	Yes	No	No	Yes	Yes
DCFL	No	No	Yes	No	No
VPL	Yes	Yes	Yes	Yes	Yes

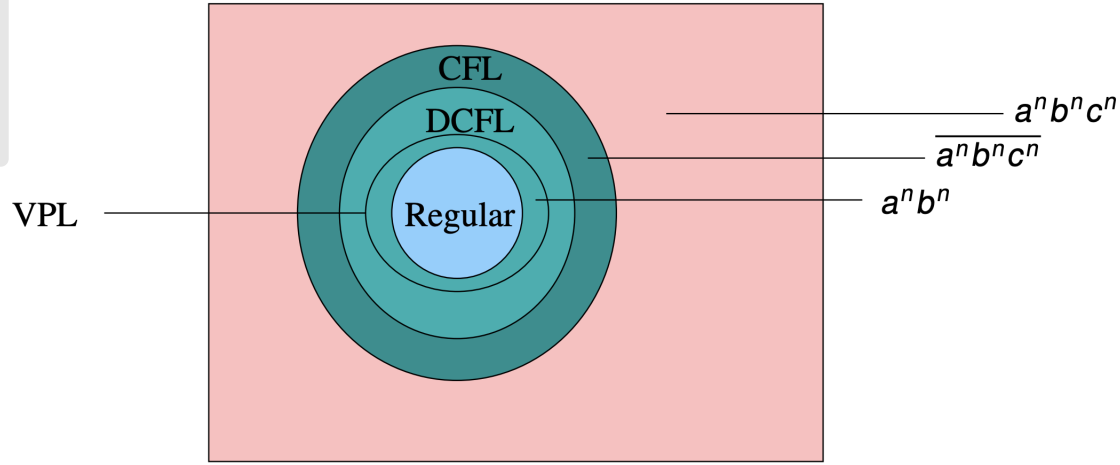
The success of Visibly Pushdown Languages (VPLs)

$\mathbb{VPL}$ = Finite Automata + Input-Driven Stack

$\Sigma = \Sigma_c \cup \Sigma_i \cup \Sigma_r$ (call + internal + returns)

Push (pop) only after reading a call (return).

- Ex1: Dyck languages
- Ex2: $c^{\#} r^{\#}$ but not $r^{\#} c^{\#}$ (for $c \in \Sigma_c, r \in \Sigma_r$)
- Ex3: Every regular language is in $\mathbb{VPL}$



Why do we care?

	Decision problems for automata		
	Emptiness	Universality/Equivalence	Inclusion
NFA	NLOGSPACE	PSPACE	PSPACE
PDA	PTIME	Undecidable	Undecidable
DPDA	PTIME	Decidable	Undecidable
VPA	PTIME	EXPTIME	EXPTIME

	Closure under				
	Union	Intersection	Complement	Concat.	Kleene-$*$
Regular	Yes	Yes	Yes	Yes	Yes
CFL	Yes	No	No	Yes	Yes
DCFL	No	No	Yes	No	No
VPL	Yes	Yes	Yes	Yes	Yes

- Verification of recursive programs

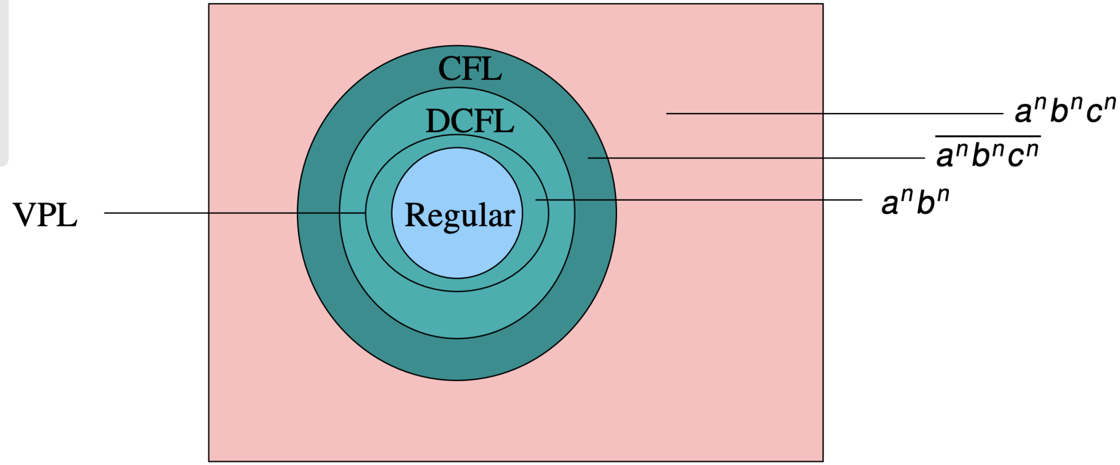
The success of Visibly Pushdown Languages (VPLs)

$\mathbb{VPL}$ = Finite Automata + Input-Driven Stack

$\Sigma = \Sigma_c \cup \Sigma_i \cup \Sigma_r$ (call + internal + returns)

Push (pop) only after reading a call (return).

- Ex1: Dyck languages
- Ex2: $c^{\#} r^{\#}$ but not $r^{\#} c^{\#}$ (for $c \in \Sigma_c, r \in \Sigma_r$)
- Ex3: Every regular language is in $\mathbb{VPL}$



Why do we care?

	Decision problems for automata		
	Emptiness	Universality/Equivalence	Inclusion
NFA	NLOGSPACE	PSPACE	PSPACE
PDA	PTIME	Undecidable	Undecidable
DPDA	PTIME	Decidable	Undecidable
VPA	PTIME	EXPTIME	EXPTIME

	Closure under				
	Union	Intersection	Complement	Concat.	Kleene-$*$
Regular	Yes	Yes	Yes	Yes	Yes
CFL	Yes	No	No	Yes	Yes
DCFL	No	No	Yes	No	No
VPL	Yes	Yes	Yes	Yes	Yes

- Verification of recursive programs
- XML schema validation

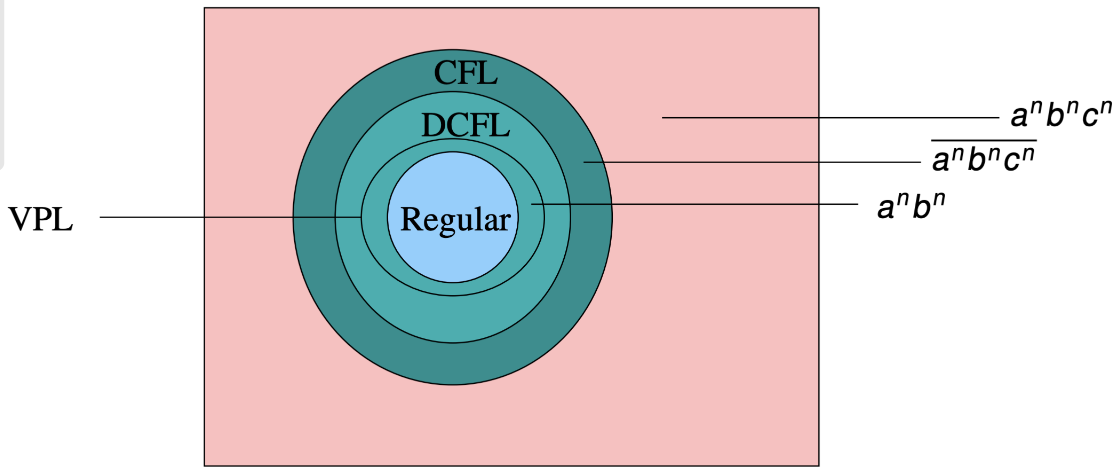
The success of Visibly Pushdown Languages (VPLs)

$\mathbb{VPL}$ = Finite Automata + Input-Driven Stack

$\Sigma = \Sigma_c \cup \Sigma_i \cup \Sigma_r$ (call + internal + returns)

Push (pop) only after reading a call (return).

- Ex1: Dyck languages
- Ex2: $c^{\#} r^{\#}$ but not $r^{\#} c^{\#}$ (for $c \in \Sigma_c, r \in \Sigma_r$)
- Ex3: Every regular language is in $\mathbb{VPL}$



Why do we care?

	Decision problems for automata		
	Emptiness	Universality/Equivalence	Inclusion
NFA	NLOGSPACE	PSPACE	PSPACE
PDA	PTIME	Undecidable	Undecidable
DPDA	PTIME	Decidable	Undecidable
VPA	PTIME	EXPTIME	EXPTIME

	Closure under				
	Union	Intersection	Complement	Concat.	Kleene-*
Regular	Yes	Yes	Yes	Yes	Yes
CFL	Yes	No	No	Yes	Yes
DCFL	No	No	Yes	No	No
VPL	Yes	Yes	Yes	Yes	Yes

- Verification of recursive programs
- XML schema validation

Why not to employ $\mathbb{VPL}$ in knowledge representation?

Is $\mathcal{ALC}_{\mathsf{vpl}}$ robust under extensions with features supported by W3C ontology languages?

Is $\mathcal{ALC}_{\mathsf{vpl}}$ robust under extensions with features supported by W3C ontology languages?

- $\mathcal{ALC}_{\mathsf{vpl}}$ is decidable and 2ExpTime-complete (Löding et. al 2007)

Is $\mathcal{ALC}_{\mathsf{vpl}}$ robust under extensions with features supported by W3C ontology languages?

- $\mathcal{ALC}_{\mathsf{vpl}}$ is decidable and $2\textsc{ExpTime}$-complete (Löding et. al 2007)
- $\mathcal{ALC}_{\mathsf{vpl}}$ is inverses is undecidable (unpublished, discovered in Stefan Göller's PhD Thesis'2008)

Is $\mathcal{ALC}_{\mathsf{vpl}}$ robust under extensions with features supported by W3C ontology languages?

- $\mathcal{ALC}_{\mathsf{vpl}}$ is decidable and 2ExpTime-complete (Löding et. al 2007)
- $\mathcal{ALC}_{\mathsf{vpl}}$ is inverses is undecidable (unpublished, discovered in Stefan Göller's PhD Thesis'2008)

How about other features? How about querying?

Is $\mathcal{ALC}_{\mathsf{vpl}}$ robust under extensions with features supported by W3C ontology languages?

- $\mathcal{ALC}_{\mathsf{vpl}}$ is decidable and 2ExpTime-complete (Löding et. al 2007)
- $\mathcal{ALC}_{\mathsf{vpl}}$ is inverses is undecidable (unpublished, discovered in Stefan Göller's PhD Thesis'2008)

How about other features? How about querying?

Beyond $\mathcal{ALC}_{\mathsf{reg}}$: Exploring Non-Regular Extensions of PDL with Description Logics Features

Bartosz Bednarczyk[1,2]

Is $\mathcal{ALC}_{\mathsf{vpl}}$ robust under extensions with features supported by W3C ontology languages?

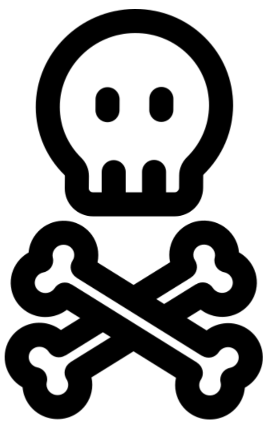
- $\mathcal{ALC}_{\mathsf{vpl}}$ is decidable and 2ExpTime-complete (Löding et. al 2007)
- $\mathcal{ALC}_{\mathsf{vpl}}$ is inverses is undecidable (unpublished, discovered in Stefan Göller's PhD Thesis'2008)

How about other features? How about querying?

Beyond $\mathcal{ALC}_{\mathsf{reg}}$: Exploring Non-Regular Extensions of PDL with Description Logics Features

Bartosz Bednarczyk[1,2]

Loops



Is $\mathcal{ALC}_{\mathsf{vpl}}$ robust under extensions with features supported by W3C ontology languages?

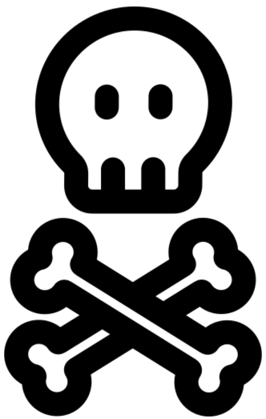
- $\mathcal{ALC}_{\mathsf{vpl}}$ is decidable and 2ExpTime-complete (Löding et. al 2007)
- $\mathcal{ALC}_{\mathsf{vpl}}$ is inverses is undecidable (unpublished, discovered in Stefan Göller's PhD Thesis'2008)

How about other features? How about querying?

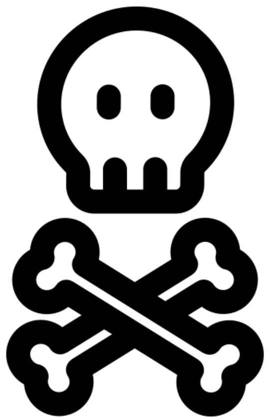
Beyond $\mathcal{ALC}_{\mathsf{reg}}$: Exploring Non-Regular Extensions of PDL with Description Logics Features

Bartosz Bednarczyk[1,2]

Loops



Nominals



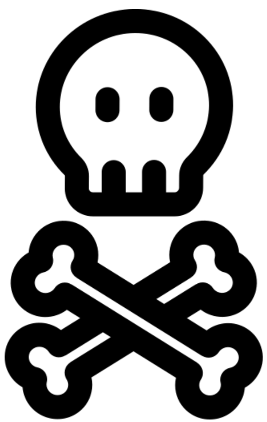
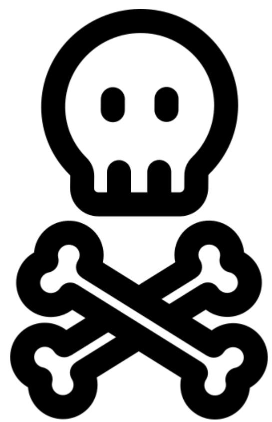
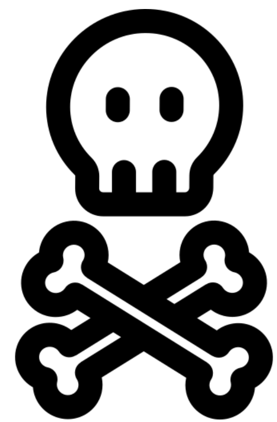
Is $\mathcal{ALC}_{\mathsf{vpl}}$ robust under extensions with features supported by W3C ontology languages?

- $\mathcal{ALC}_{\mathsf{vpl}}$ is decidable and 2EXPTIME-complete (Löding et. al 2007)
- $\mathcal{ALC}_{\mathsf{vpl}}$ is inverses is undecidable (unpublished, discovered in Stefan Göller's PhD Thesis'2008)

How about other features? How about querying?

Beyond $\mathcal{ALC}_{\mathsf{reg}}$: Exploring Non-Regular Extensions of PDL with Description Logics Features

Bartosz Bednarczyk[1,2]

Loops	Nominals	Queries
		

Is $\mathcal{ALC}_{\mathsf{vpl}}$ robust under extensions with features supported by W3C ontology languages?

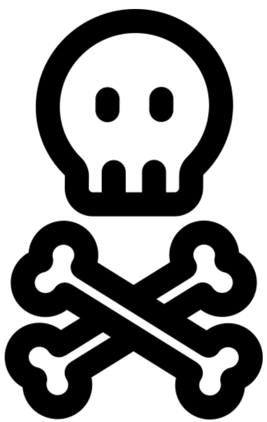
- $\mathcal{ALC}_{\mathsf{vpl}}$ is decidable and 2ExpTime-complete (Löding et. al 2007)
- $\mathcal{ALC}_{\mathsf{vpl}}$ is inverses is undecidable (unpublished, discovered in Stefan Göller's PhD Thesis'2008)

How about other features? How about querying?

Beyond $\mathcal{ALC}_{\mathsf{reg}}$: Exploring Non-Regular Extensions of PDL with Description Logics Features

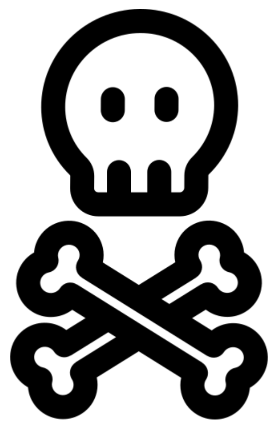
Bartosz Bednarczyk[1,2]

Loops

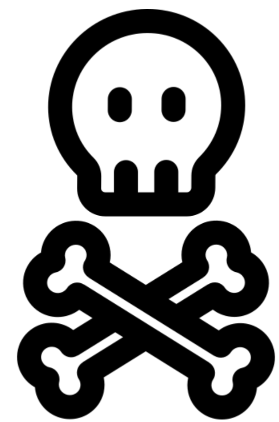


Visibly one counter

Nominals



Queries



Is $\mathcal{ALC}_{\mathsf{vpl}}$ robust under extensions with features supported by W3C ontology languages?

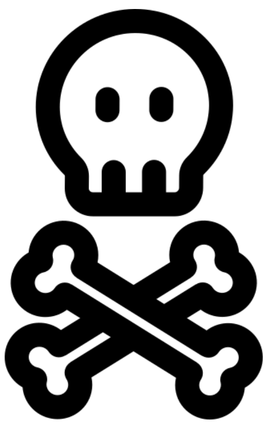
- $\mathcal{ALC}_{\mathsf{vpl}}$ is decidable and 2ExpTime-complete (Löding et. al 2007)
- $\mathcal{ALC}_{\mathsf{vpl}}$ is inverses is undecidable (unpublished, discovered in Stefan Göller's PhD Thesis'2008)

How about other features? How about querying?

Beyond $\mathcal{ALC}_{\mathsf{reg}}$: Exploring Non-Regular Extensions of PDL with Description Logics Features

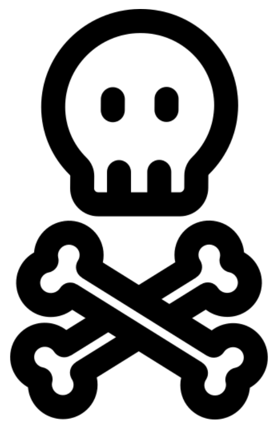
Bartosz Bednarczyk[1,2]

Loops



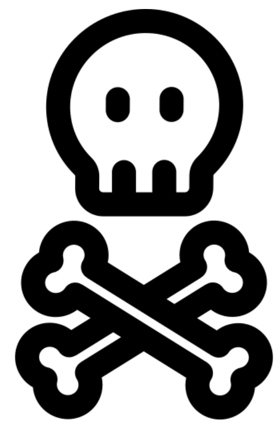
Visibly one counter

Nominals



$\mathcal{ALC}^{r\#s\#}_{\mathsf{reg}}$

Queries



Is $\mathcal{ALC}_{\mathsf{vpl}}$ robust under extensions with features supported by W3C ontology languages?

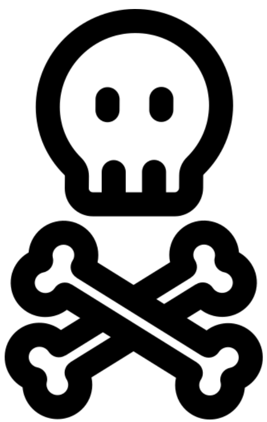
- $\mathcal{ALC}_{\mathsf{vpl}}$ is decidable and 2EXPTIME-complete (Löding et. al 2007)
- $\mathcal{ALC}_{\mathsf{vpl}}$ is inverses is undecidable (unpublished, discovered in Stefan Göller's PhD Thesis'2008)

How about other features? How about querying?

Beyond $\mathcal{ALC}_{\mathsf{reg}}$: Exploring Non-Regular Extensions of PDL with Description Logics Features

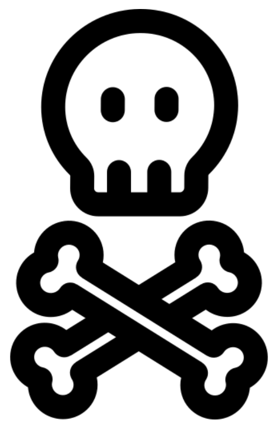
Bartosz Bednarczyk[1,2]

Loops



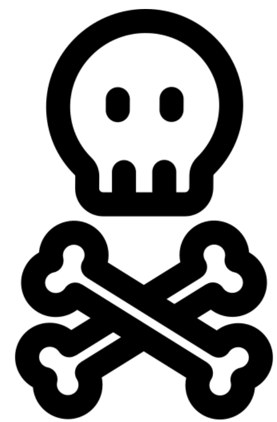
Visibly one counter

Nominals



$\mathcal{ALC}_{\mathsf{reg}}^{r^{\#}s^{\#}}$

Queries



$\mathcal{ALC}$-TBoxes + CRPQs with $r^{\#}s^{\#}$

Proof sketch: Undecidability of $\mathcal{ALC}_{\mathsf{vpl}}$ + Self

Proof sketch: Undecidability of $\mathcal{ALC}_{\mathsf{vpl}}$ + Self

Input: Deterministic one counter automata $\mathcal{A}_1$, $\mathcal{A}_2$.

Proof sketch: Undecidability of $\mathcal{ALC}_{\mathsf{vpl}}$ + Self

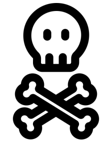
Input: Deterministic one counter automata $\mathcal{A}_1$, $\mathcal{A}_2$.

Output: Is $\mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$ non-empty?

Proof sketch: Undecidability of $\mathcal{ALC}_{\mathsf{vpl}}$ + Self

Input: Deterministic one counter automata $\mathcal{A}_1$, $\mathcal{A}_2$.

Output: Is $\mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$ non-empty?



Valiant 1973

Proof sketch: Undecidability of $\mathcal{ALC}_{\mathsf{vpl}}$ + Self

Input: Deterministic one counter automata $\mathcal{A}_1$, $\mathcal{A}_2$.

Output: Is $\mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$ non-empty?

Valiant 1973

Key insight: Deterministic one-counter languages can be projectively recognized by VPA.

Proof sketch: Undecidability of $\mathcal{ALC}_{\mathsf{vpl}}$ + Self

Input: Deterministic one counter automata $\mathcal{A}_1$, $\mathcal{A}_2$.

Output: Is $\mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$ non-empty?

Valiant 1973

Key insight: Deterministic one-counter languages can be projectively recognized by VPA.

Lemma 15. *For any DOCA $\mathcal{A}$ over Σ, we can construct a VOCA $\tilde{\mathcal{A}}$ over $\tilde{\Sigma} := (\Sigma \times \{c\}, (\Sigma \times \{i\}) \cup \{\mathtt{x}\}, \Sigma \times \{r\})$ where $\mathtt{x}$ is a fresh internal letter, such that all words in $\mathcal{L}(\mathcal{A})$ have the form $\tilde{\mathtt{a}}_1 \mathtt{x} \tilde{\mathtt{a}}_2 \mathtt{x} \ldots \mathtt{x} \tilde{\mathtt{a}}_n$ for $\tilde{\mathtt{a}}_1, \ldots, \tilde{\mathtt{a}}_n \in \Sigma \times \{c, i, r\}$, and $\mathcal{L}(\mathcal{A}) = \{\pi_1(\tilde{\mathtt{w}}) \mid \tilde{\mathtt{w}} := \tilde{\mathtt{a}}_1 \ldots \tilde{\mathtt{a}}_n,\ \tilde{\mathtt{a}}_1 \mathtt{x} \ldots \mathtt{x} \tilde{\mathtt{a}}_n \in \mathcal{L}(\tilde{\mathcal{A}})\}$ holds.*

Proof sketch: Undecidability of $\mathcal{ALC}_{\mathsf{vpl}}$ + Self

Input: Deterministic one counter automata $\mathcal{A}_1$, $\mathcal{A}_2$.

Output: Is $\mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$ non-empty?

Valiant 1973

Key insight: Deterministic one-counter languages can be projectively recognized by VPA.

Lemma 15. *For any DOCA $\mathcal{A}$ over Σ, we can construct a VOCA $\tilde{\mathcal{A}}$ over $\tilde{\Sigma} := (\Sigma \times \{c\}, (\Sigma \times \{i\}) \cup \{\mathtt{x}\}, \Sigma \times \{r\})$ where $\mathtt{x}$ is a fresh internal letter, such that all words in $\mathcal{L}(\mathcal{A})$ have the form $\tilde{\mathtt{a}_1}\mathtt{x}\tilde{\mathtt{a}_2}\mathtt{x}\ldots\mathtt{x}\tilde{\mathtt{a}_n}$ for $\tilde{\mathtt{a}_1}, \ldots, \tilde{\mathtt{a}_n} \in \Sigma \times \{c, i, r\}$, and $\mathcal{L}(\mathcal{A}) = \{\pi_1(\tilde{\mathtt{w}}) \mid \tilde{\mathtt{w}} := \tilde{\mathtt{a}_1}\ldots\tilde{\mathtt{a}_n},\ \tilde{\mathtt{a}_1}\mathtt{x}\ldots\mathtt{x}\tilde{\mathtt{a}_n} \in \mathcal{L}(\tilde{\mathcal{A}})\}$ holds.*

Given DOCA $\mathcal{A}_1, \mathcal{A}_2$, we get VPA $\hat{\mathcal{A}}_1, \hat{\mathcal{A}}_2$ projectively recognizing their lang. + $\hat{\mathcal{C}}_1, \hat{\mathcal{C}}_2$ for complements

Proof sketch: Undecidability of $\mathcal{ALC}_{\mathsf{vpl}}$ + Self

Input: Deterministic one counter automata $\mathcal{A}_1$, $\mathcal{A}_2$.

Output: Is $\mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$ non-empty?

Valiant 1973

Key insight: Deterministic one-counter languages can be projectively recognized by VPA.

Lemma 15. *For any DOCA $\mathcal{A}$ over Σ, we can construct a VOCA $\tilde{\mathcal{A}}$ over $\underline{\tilde{\Sigma} := (\Sigma \times \{c\}, (\Sigma \times \{i\}) \cup \{\mathtt{x}\}, \Sigma \times \{r\})}$ where $\mathtt{x}$ is a fresh internal letter, such that all words in $\mathcal{L}(\mathcal{A})$ have the form $\tilde{\mathtt{a}_1}\mathtt{x}\tilde{\mathtt{a}_2}\mathtt{x} \ldots \mathtt{x}\tilde{\mathtt{a}_n}$ for $\tilde{\mathtt{a}_1}, \ldots, \tilde{\mathtt{a}_n} \in \Sigma \times \{c, i, r\}$, and $\mathcal{L}(\mathcal{A}) = \{\pi_1(\tilde{\mathtt{w}}) \mid \tilde{\mathtt{w}} := \tilde{\mathtt{a}_1} \ldots \tilde{\mathtt{a}_n},\ \tilde{\mathtt{a}_1}\mathtt{x} \ldots \mathtt{x}\tilde{\mathtt{a}_n} \in \mathcal{L}(\tilde{\mathcal{A}})\}$ holds.*

Given DOCA $\mathcal{A}_1, \mathcal{A}_2$, we get VPA $\hat{\mathcal{A}}_1, \hat{\mathcal{A}}_2$ projectively recognizing their lang. $+$ $\hat{\mathcal{C}}_1, \hat{\mathcal{C}}_2$ for complements

Trick 1: Encode "word-like structures" with loops storing the actual letters.

Proof sketch: Undecidability of $\mathcal{ALC}_{\mathsf{vpl}}$ + Self

Input: Deterministic one counter automata $\mathcal{A}_1$, $\mathcal{A}_2$.

Output: Is $\mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$ non-empty?

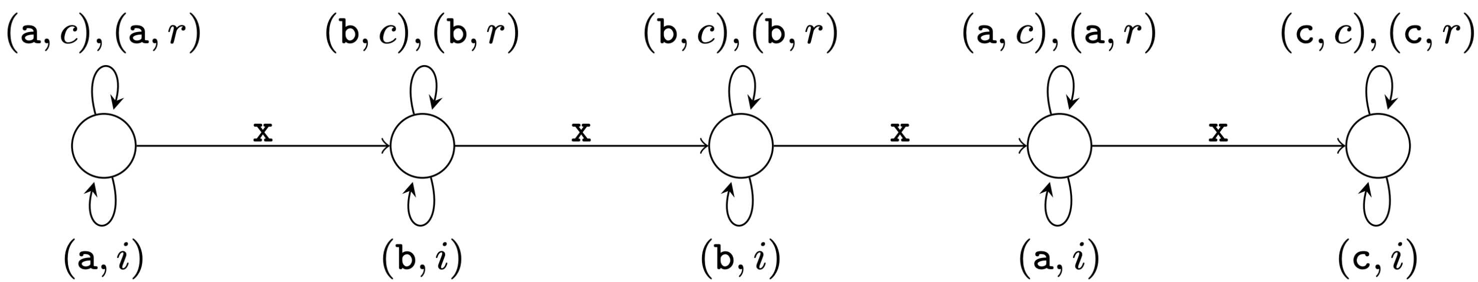
Valiant 1973

Key insight: Deterministic one-counter languages can be projectively recognized by VPA.

Lemma 15. *For any DOCA $\mathcal{A}$ over Σ, we can construct a VOCA $\tilde{\mathcal{A}}$ over $\underline{\tilde{\Sigma} \coloneqq (\Sigma \times \{c\}, (\Sigma \times \{i\}) \cup \{\mathtt{x}\}, \Sigma \times \{r\})}$ where $\mathtt{x}$ is a fresh internal letter, such that all words in $\mathcal{L}(\tilde{\mathcal{A}})$ have the form $\tilde{\mathtt{a}_1}\mathtt{x}\tilde{\mathtt{a}_2}\mathtt{x}\ldots\mathtt{x}\tilde{\mathtt{a}_n}$ for $\tilde{\mathtt{a}_1}, \ldots, \tilde{\mathtt{a}_n} \in \Sigma \times \{c, i, r\}$, and $\mathcal{L}(\mathcal{A}) = \{\pi_1(\tilde{\mathtt{w}}) \mid \tilde{\mathtt{w}} \coloneqq \tilde{\mathtt{a}_1}\ldots\tilde{\mathtt{a}_n},\ \tilde{\mathtt{a}_1}\mathtt{x}\ldots\mathtt{x}\tilde{\mathtt{a}_n} \in \mathcal{L}(\tilde{\mathcal{A}})\}$ holds.*

Given DOCA $\mathcal{A}_1, \mathcal{A}_2$, we get VPA $\hat{\mathcal{A}}_1, \hat{\mathcal{A}}_2$ projectively recognizing their lang. + $\hat{\mathcal{C}}_1, \hat{\mathcal{C}}_2$ for complements

Trick 1: Encode "word-like structures" with loops storing the actual letters. Example: abbac



Input: Deterministic one counter automata $\mathcal{A}_1$, $\mathcal{A}_2$.

Output: Is $\mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$ non-empty?

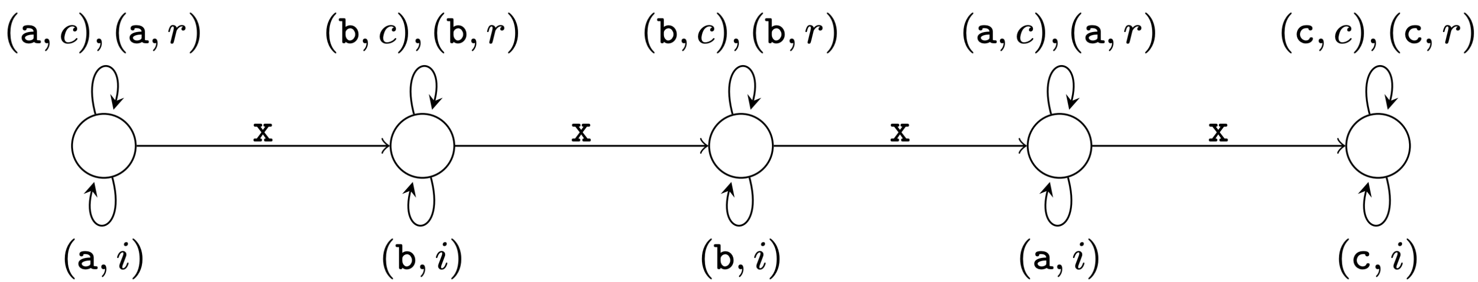
Valiant 1973

Key insight: Deterministic one-counter languages can be projectively recognized by VPA.

Lemma 15. *For any DOCA $\mathcal{A}$ over Σ, we can construct a VOCA $\tilde{\mathcal{A}}$ over $\underline{\tilde{\Sigma} \coloneqq (\Sigma \times \{c\}, (\Sigma \times \{i\}) \cup \{\mathtt{x}\}, \Sigma \times \{r\})}$ where $\mathtt{x}$ is a fresh internal letter, such that all words in $\mathcal{L}(\tilde{\mathcal{A}})$ have the form $\tilde{\mathtt{a}_1}\mathtt{x}\tilde{\mathtt{a}_2}\mathtt{x}\ldots\mathtt{x}\tilde{\mathtt{a}_n}$ for $\tilde{\mathtt{a}_1}, \ldots, \tilde{\mathtt{a}_n} \in \Sigma \times \{c, i, r\}$, and $\mathcal{L}(\mathcal{A}) = \{\pi_1(\tilde{\mathtt{w}}) \mid \tilde{\mathtt{w}} \coloneqq \tilde{\mathtt{a}_1} \ldots \tilde{\mathtt{a}_n},\ \tilde{\mathtt{a}_1}\mathtt{x} \ldots \mathtt{x}\tilde{\mathtt{a}_n} \in \mathcal{L}(\tilde{\mathcal{A}})\}$ holds.*

Given DOCA $\mathcal{A}_1, \mathcal{A}_2$, we get VPA $\hat{\mathcal{A}}_1, \hat{\mathcal{A}}_2$ projectively recognizing their lang. + $\hat{\mathcal{C}}_1, \hat{\mathcal{C}}_2$ for complements

Trick 1: Encode "word-like structures" with loops storing the actual letters. Example: abbac



Trick 2: Employ concepts $\forall\hat{\mathcal{A}}_1.\mathrm{OK}_1 \sqcap \forall\hat{\mathcal{C}}_1.\neg\mathrm{OK}_1$ to decorate interpretations with "acceptance" of $\mathcal{A}_1$.

Proof sketch: Undecidability of $\mathcal{ALCO} + r^{\#}s^{\#}$ (Introduction)

Proof sketch: Undecidability of $\mathcal{ALCO} + r^{\#}s^{\#}$ (Introduction)

Input: A finite set of 4-sided tiles with a distinguished colour $\square$.

Proof sketch: Undecidability of $\mathcal{ALCO} + r^{\#}s^{\#}$ (Introduction)

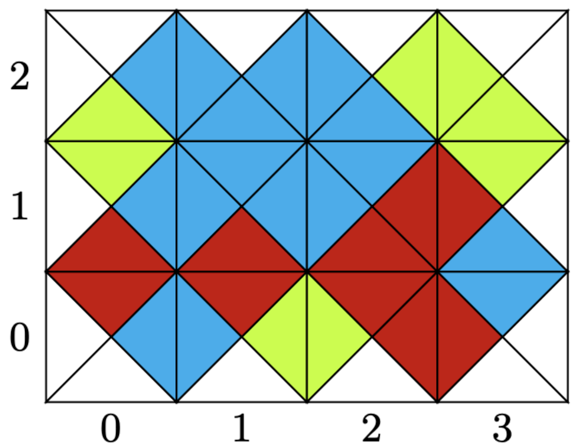
Input: A finite set of 4-sided tiles with a distinguished colour $\square$.

Output: Is there $\mathrm{N}, \mathrm{M} \in \mathbb{N}$ so that we can cover a $\square$-bordered $(\mathrm{N} \times \mathrm{M})$ rectangle w.r.t tiling rules?

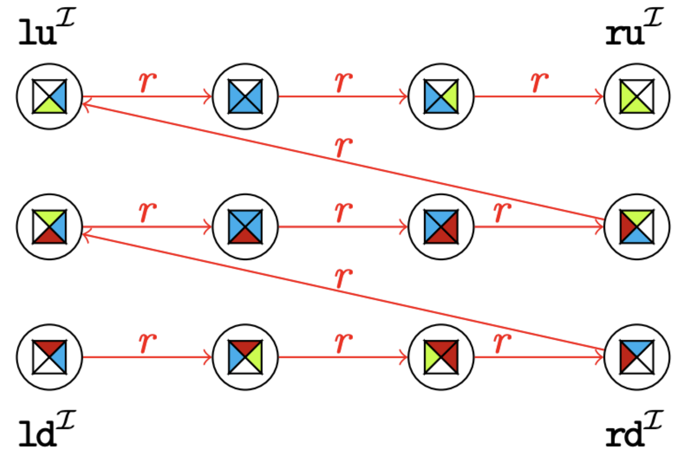
Proof sketch: Undecidability of $\mathcal{ALCO} + r^{\#}s^{\#}$ (Introduction)

Input: A finite set of 4-sided tiles with a distinguished colour $\square$.

Output: Is there $\mathrm{N}, \mathrm{M} \in \mathbb{N}$ so that we can cover a $\square$-bordered $(\mathrm{N} \times \mathrm{M})$ rectangle w.r.t tiling rules?



(a) Visualization of ξ.



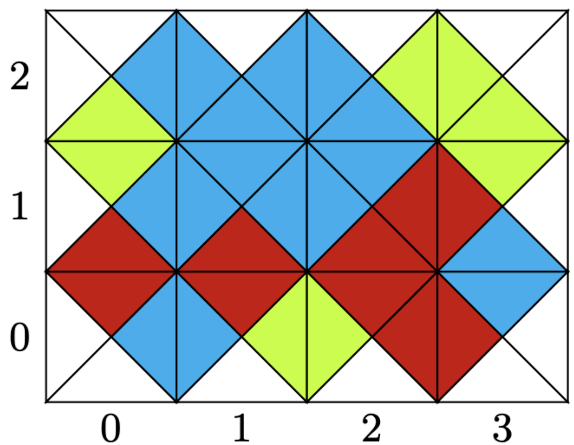
(b) The encoding of ξ as a $\mathcal{D}$-snake $\mathcal{I}$.

Fig. 1: If $\mathrm{Col} = \{$, , , $\}$ and $\mathrm{T} = \mathrm{Col}^4$, the map $\xi := \{(0,0) \mapsto$, $(1,0) \mapsto$, $(2,0) \mapsto$, $(3,0) \mapsto$, $(0,1) \mapsto$, $(1,1) \mapsto$, $(2,1) \mapsto$, $(3,1) \mapsto$, $(0,2) \mapsto$, $(1,2) \mapsto$, $(2,2) \mapsto$, $(3,2) \mapsto$ $\}$ covers $\mathbb{Z}_4 \times \mathbb{Z}_3$.

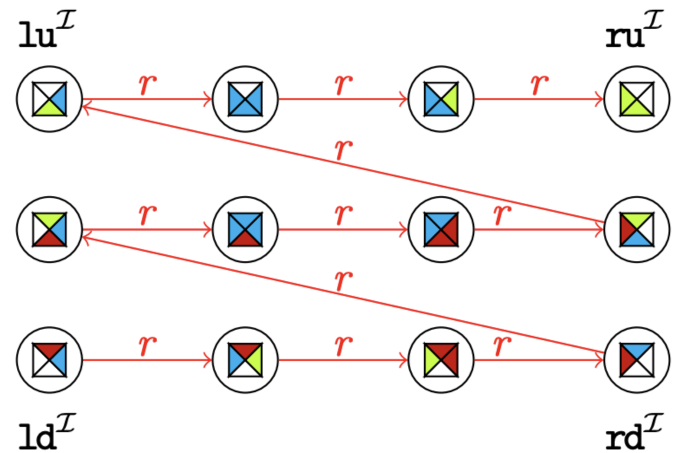
Proof sketch: Undecidability of $\mathcal{ALCO} + r^{\#}s^{\#}$ (Introduction)

Input: A finite set of 4-sided tiles with a distinguished colour □.

Output: Is there $\mathrm{N}, \mathrm{M} \in \mathbb{N}$ so that we can cover a □-bordered $(\mathrm{N} \times \mathrm{M})$ rectangle w.r.t tiling rules?



(a) Visualization of ξ.



(b) The encoding of ξ as a $\mathscr{D}$-snake $\mathcal{I}$.

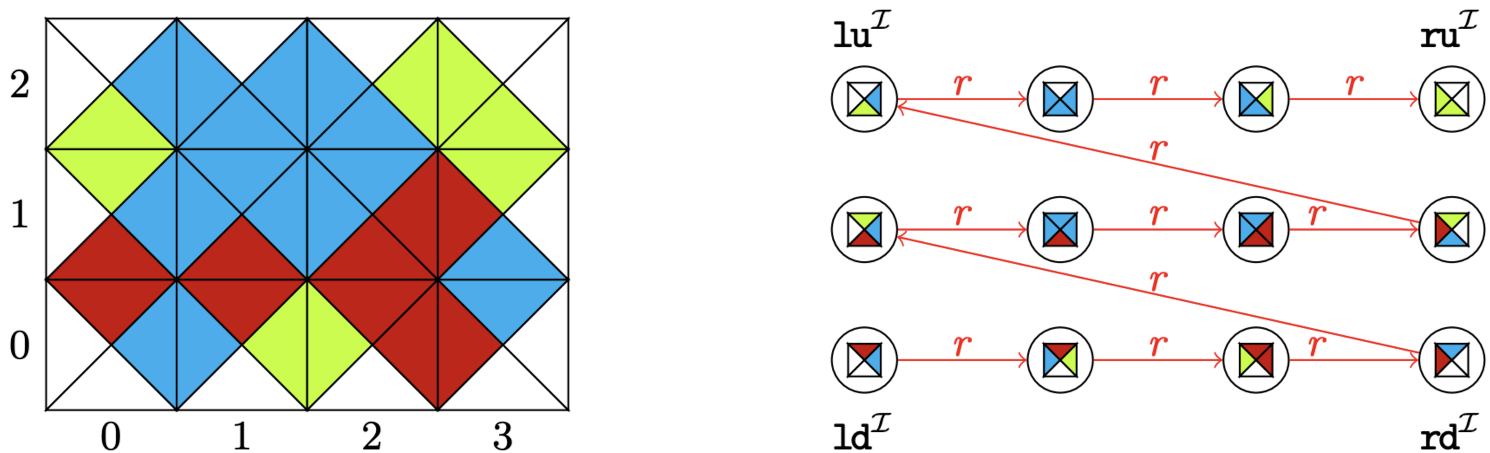
Fig. 1: If Col $= \{$■, ■, □, ■$\}$ and $\mathrm{T} = \mathrm{Col}^4$, the map $\xi := \{(0,0) \mapsto$ ⊠, $(1,0) \mapsto$ ⊠, $(2,0) \mapsto$ ⊠, $(3,0) \mapsto$ ⊠, $(0,1) \mapsto$ ⊠, $(1,1) \mapsto$ ⊠, $(2,1) \mapsto$ ⊠, $(3,1) \mapsto$ ⊠, $(0,2) \mapsto$ ⊠, $(1,2) \mapsto$ ⊠, $(2,2) \mapsto$ ⊠, $(3,2) \mapsto$ ⊠$\}$ covers $\mathbb{Z}_4 \times \mathbb{Z}_3$.

Problem 1: How to express existence of an N such that every N steps from the start a left-border tile occurs?

Proof sketch: Undecidability of $\mathcal{ALCO} + r^{\#}s^{\#}$ (Introduction)

Input: A finite set of 4-sided tiles with a distinguished colour $\square$.

Output: Is there $\mathrm{N}, \mathrm{M} \in \mathbb{N}$ so that we can cover a $\square$-bordered $(\mathrm{N} \times \mathrm{M})$ rectangle w.r.t tiling rules?



(a) Visualization of ξ.

(b) The encoding of ξ as a $\mathcal{D}$-snake $\mathcal{I}$.

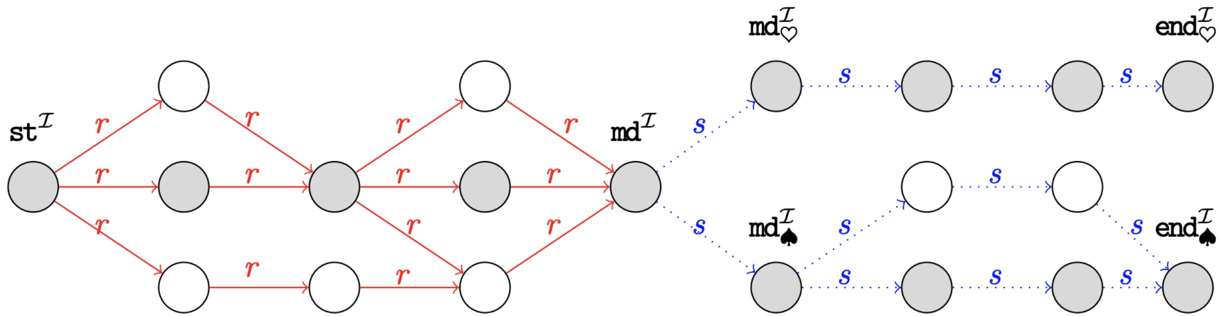
Fig. 1: If $\mathrm{Col} = \{\square, \square, \square, \square\}$ and $\mathrm{T} = \mathrm{Col}^4$, the map $\xi := \{(0,0) \mapsto \boxtimes, (1,0) \mapsto \boxtimes, (2,0) \mapsto \boxtimes, (3,0) \mapsto \boxtimes, (0,1) \mapsto \boxtimes, (1,1) \mapsto \boxtimes, (2,1) \mapsto \boxtimes, (3,1) \mapsto \boxtimes, (0,2) \mapsto \boxtimes, (1,2) \mapsto \boxtimes, (2,2) \mapsto \boxtimes, (3,2) \mapsto \boxtimes\}$ covers $\mathbb{Z}_4 \times \mathbb{Z}_3$.

Problem 1: How to express existence of an N such that every N steps from the start a left-border tile occurs?

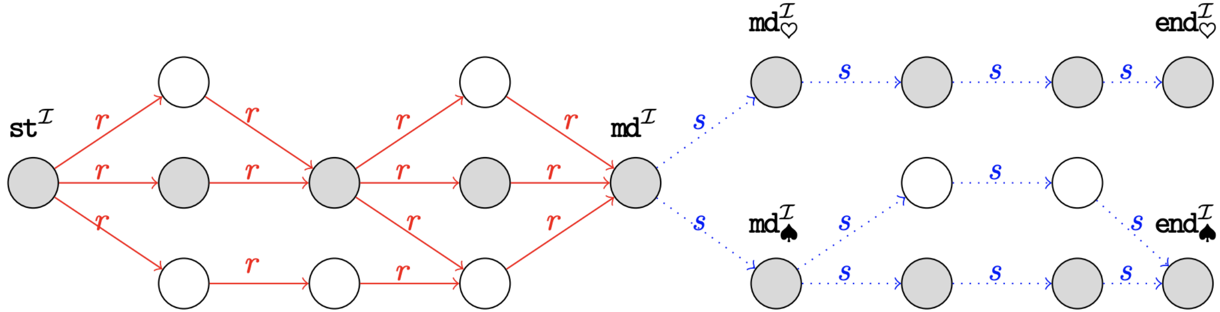
Problem 2: How to express that a tile and the tile N steps further have matching sides?

To solve problems from the previous slide, we must teach snakes how to measure. Use yardsticks!

To solve problems from the previous slide, we must teach snakes how to measure. Use yardsticks!

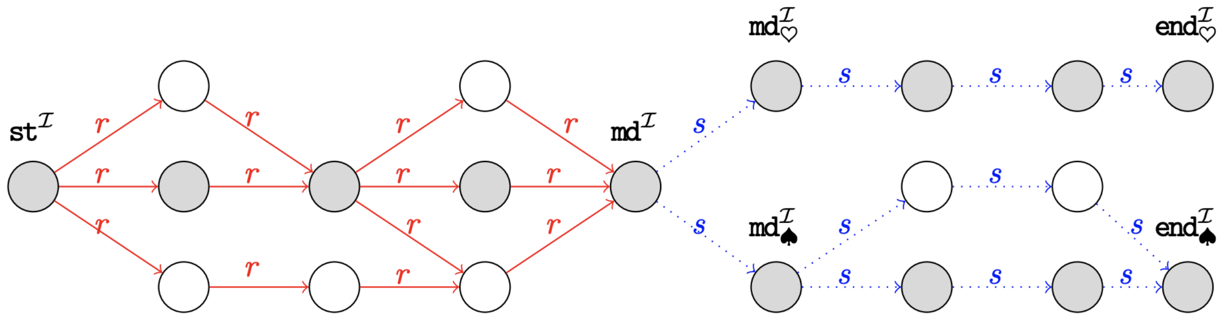


To solve problems from the previous slide, we must teach snakes how to measure. Use yardsticks!



Key property: there is unique N s.t. distances $\mathtt{st} \rightsquigarrow \mathtt{md}$ and $\mathtt{md} \rightsquigarrow \mathtt{end}_{\mathrm{t}}$ are all equal to N.

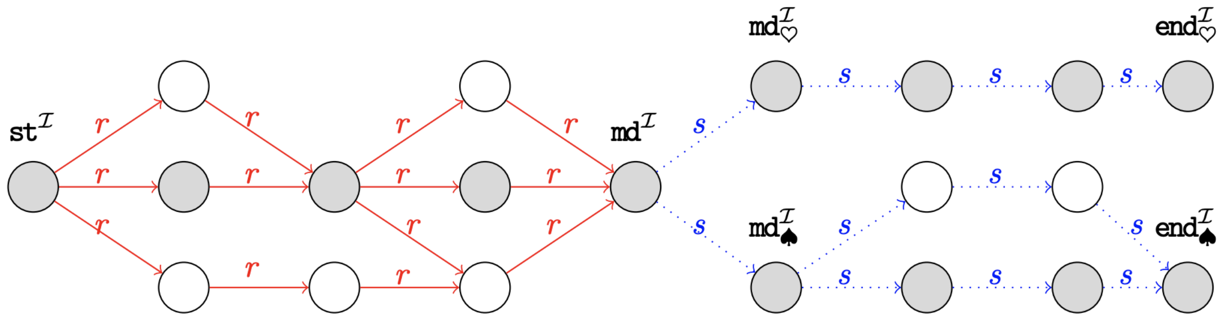
To solve problems from the previous slide, we must teach snakes how to measure. Use yardsticks!



Key property: there is unique N s.t. distances $\mathtt{st} \rightsquigarrow \mathtt{md}$ and $\mathtt{md} \rightsquigarrow \mathtt{end}_{\mathrm{t}}$ are all equal to N.

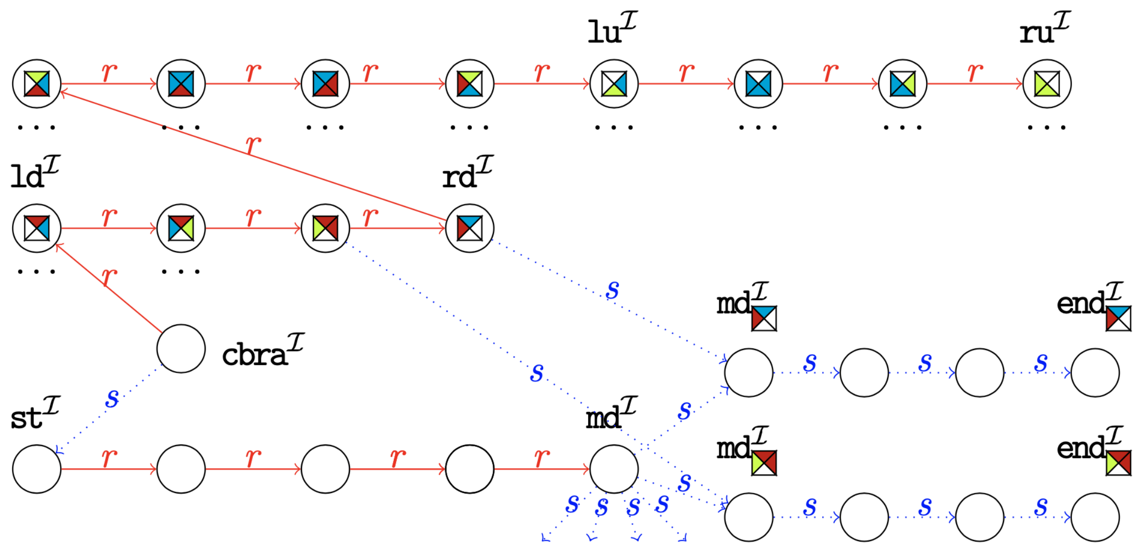
We sychronize snakes and yardsticks obtaining metricobras. Metricobras exist iff tiling systems are solvable.

To solve problems from the previous slide, we must teach snakes how to measure. Use yardsticks!

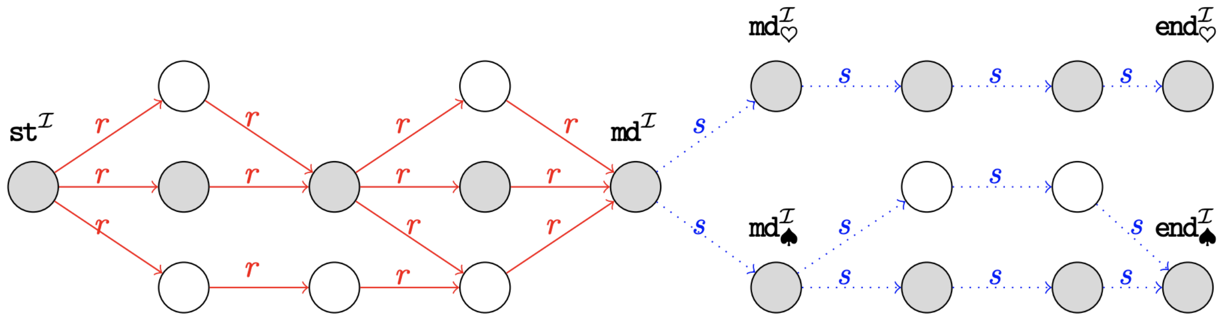


Key property: there is unique N s.t. distances $\mathtt{st} \rightsquigarrow \mathtt{md}$ and $\mathtt{md} \rightsquigarrow \mathtt{end}_{\mathrm{t}}$ are all equal to N.

We sychronize snakes and yardsticks obtaining metricobras. Metricobras exist iff tiling systems are solvable.



To solve problems from the previous slide, we must teach snakes how to measure. Use yardsticks!

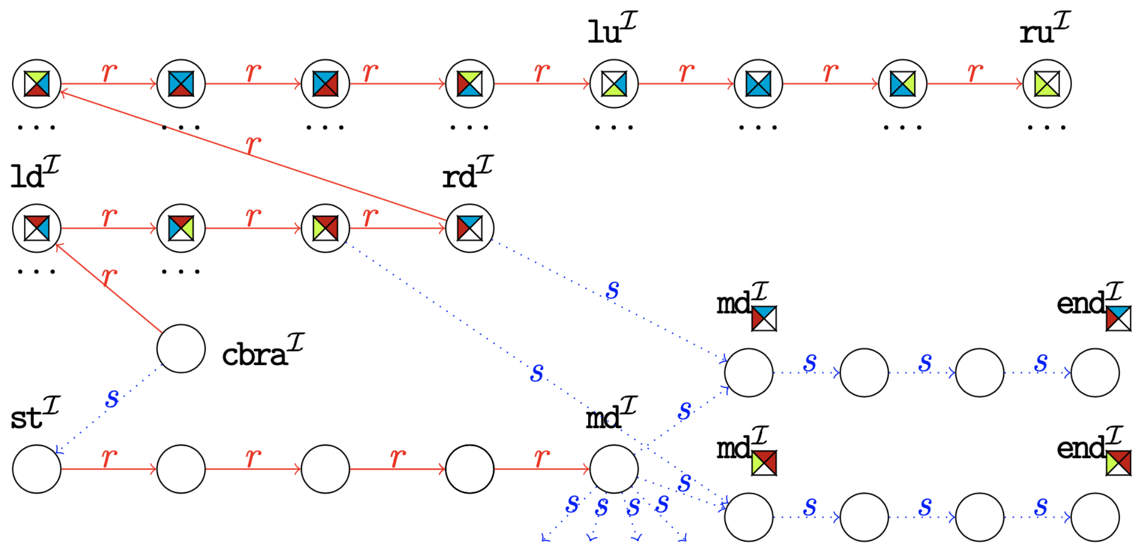


Key property: there is unique N s.t. distances $\mathtt{st} \rightsquigarrow \mathtt{md}$ and $\mathtt{md} \rightsquigarrow \mathtt{end}_{\mathrm{t}}$ are all equal to N.

We sychronize snakes and yardsticks obtaining metricobras. Metricobras exist iff tiling systems are solvable.

Key property:

An element N steps after d carries a tile t iff d can $r^{\#}s^{\#}$ reach $\mathrm{end}_{\mathrm{t}}$.



Proof sketch: Undecidability of querying $\mathcal{ALC}$-TBoxes with non-regular queries

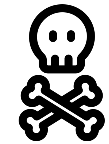
Proof sketch: Undecidability of querying $\mathcal{ALC}$-TBoxes with non-regular queries

Input: A finite set of 4-sided tiles with a distinguished colour $\square$.

Proof sketch: Undecidability of querying $\mathcal{ALC}$-TBoxes with non-regular queries

Input: A finite set of 4-sided tiles with a distinguished colour $\square$.

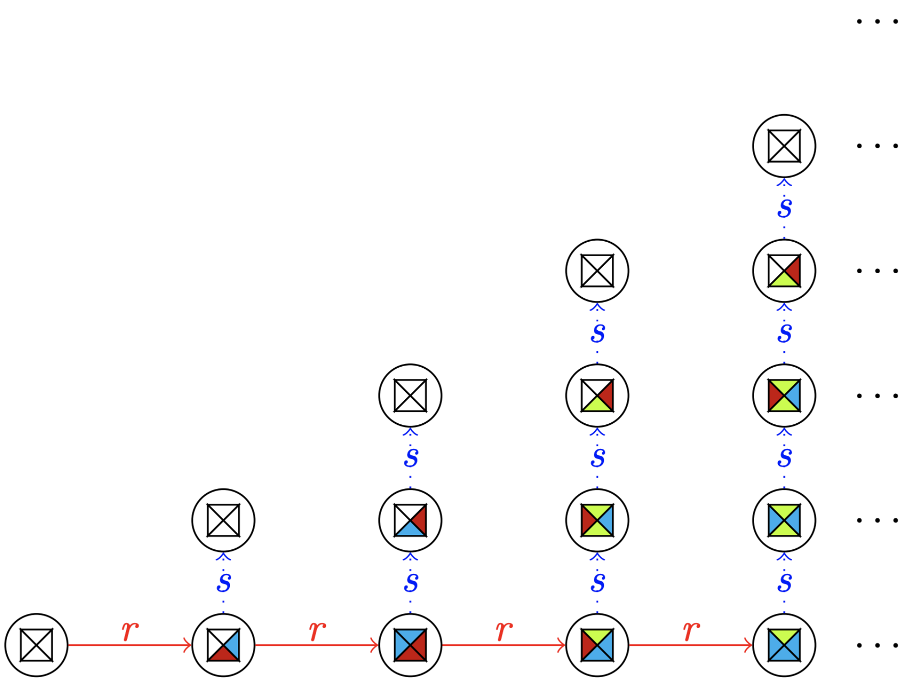
Output: Can we cover an infinite triangle (a.k.a. octant) according to tiling rules?



Proof sketch: Undecidability of querying $\mathcal{ALC}$-TBoxes with non-regular queries

Input: A finite set of 4-sided tiles with a distinguished colour $\square$.

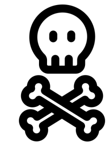
Output: Can we cover an infinite triangle (a.k.a. octant) according to tiling rules?



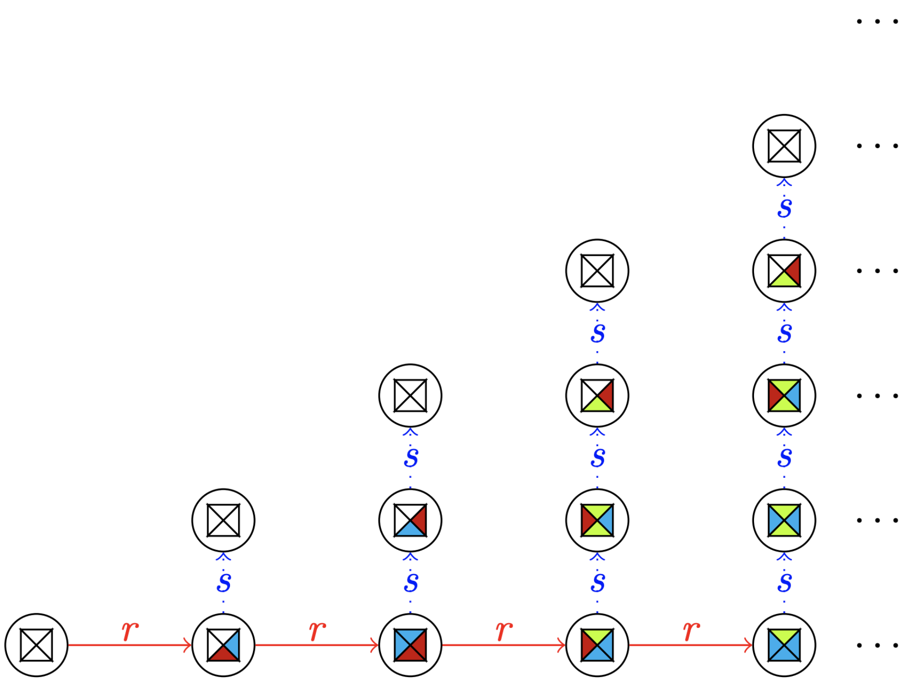
Proof sketch: Undecidability of querying $\mathcal{ALC}$-TBoxes with non-regular queries

Input: A finite set of 4-sided tiles with a distinguished colour □.

Output: Can we cover an infinite triangle (a.k.a. octant) according to tiling rules?



Proof idea: the ontology defines "octant-like" models and the query detects errors with tiling.

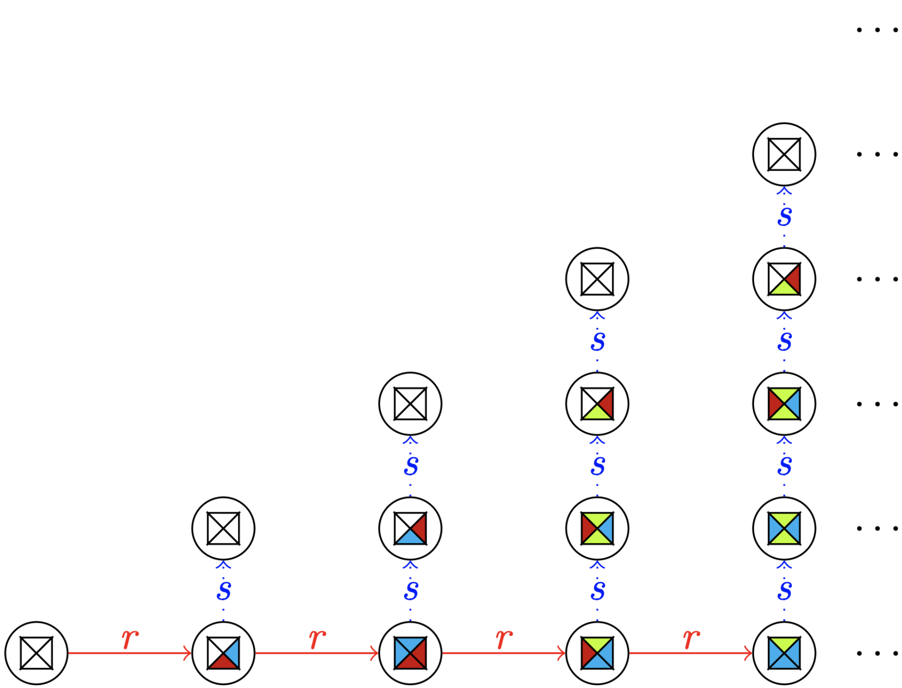


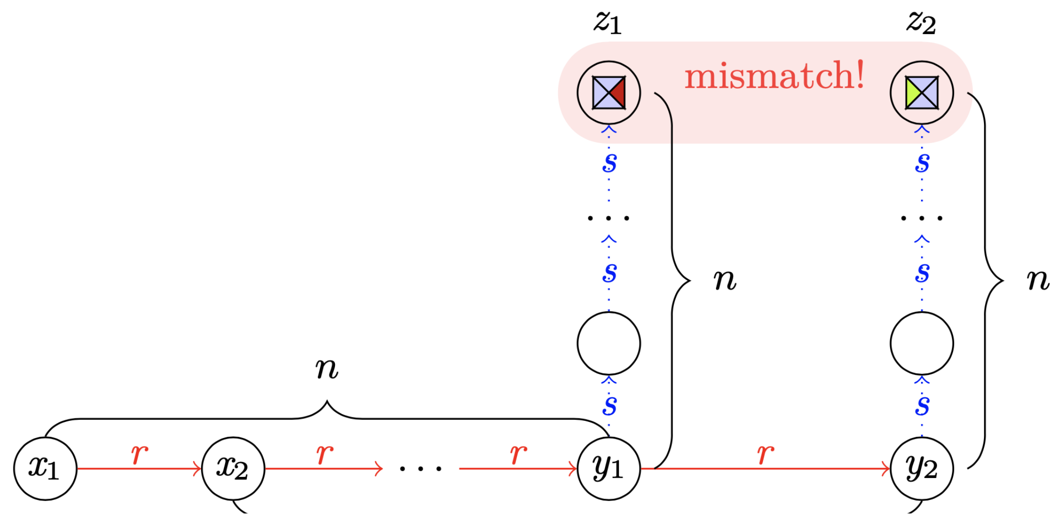
Proof sketch: Undecidability of querying $\mathcal{ALC}$-TBoxes with non-regular queries

Input: A finite set of 4-sided tiles with a distinguished colour $\square$.

Output: Can we cover an infinite triangle (a.k.a. octant) according to tiling rules?

Proof idea: the ontology defines "octant-like" models and the query detects errors with tiling.





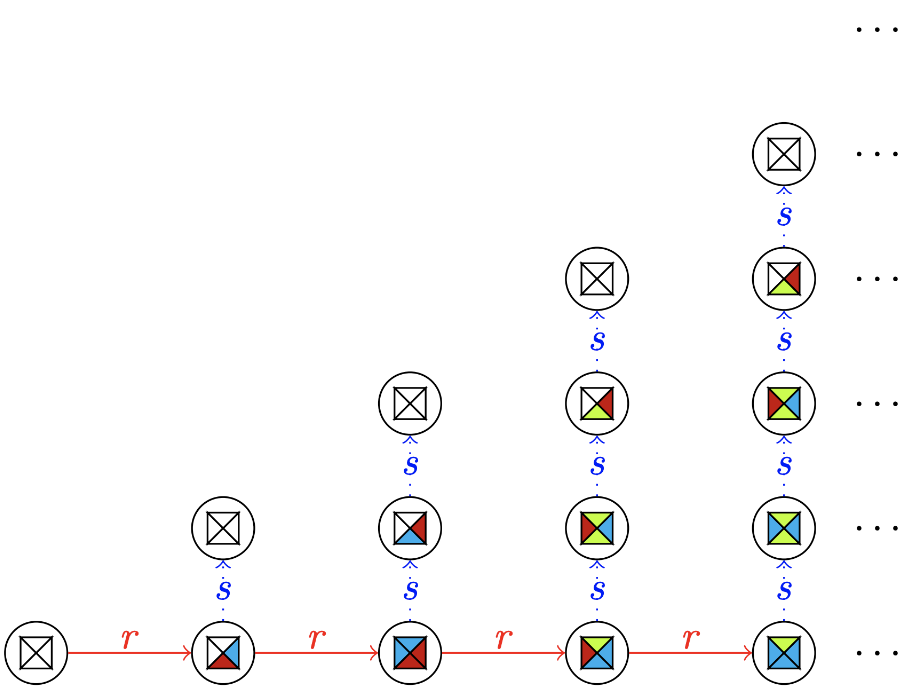
$$q^{\mathcal{D}}_{\blacktriangle} := \bigvee_{\mathrm{t},\mathrm{t}' \text{ violating (OHori)}} \big[r(x_1,x_2) \wedge r^*(x_2,y_1) \wedge r(y_1,y_2) \wedge s^*(y_1,z_1) \wedge s^*(y_2,z_2)$$
$$\wedge\, r^\# s^\#(x_1,z_1) \wedge r^\# s^\#(x_2,z_2) \wedge \mathrm{C}_{\mathrm{t}}(z_1) \wedge \mathrm{C}_{\mathrm{t}'}(z_2)\big]$$

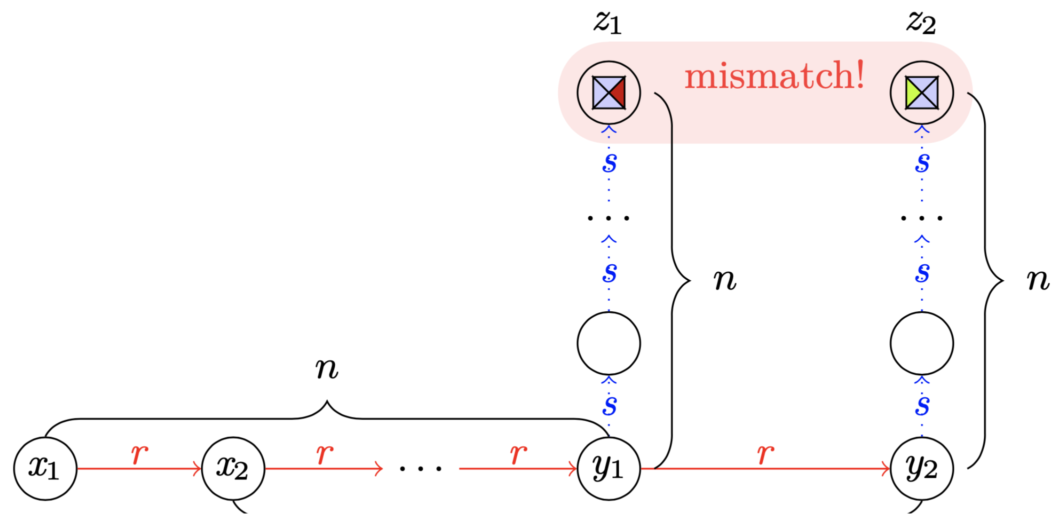
Proof sketch: Undecidability of querying $\mathcal{ALC}$-TBoxes with non-regular queries

Input: A finite set of 4-sided tiles with a distinguished colour $\square$.

Output: Can we cover an infinite triangle (a.k.a. octant) according to tiling rules?

Proof idea: the ontology defines "octant-like" models and the query detects errors with tiling.





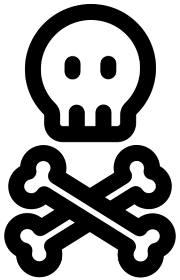
$$q^{\mathcal{D}}_{\blacktriangle} := \bigvee_{t,t' \text{ violating (OHori)}} \big[r(x_1,x_2) \wedge r^*(x_2,y_1) \wedge r(y_1,y_2) \wedge s^*(y_1,z_1) \wedge s^*(y_2,z_2)$$
$$\wedge\, r^{\#}s^{\#}(x_1,z_1) \wedge r^{\#}s^{\#}(x_2,z_2) \wedge \mathrm{C}_{t}(z_1) \wedge \mathrm{C}_{t'}(z_2)\big]$$

Key Property: $\mathrm{C} \not\models q$ iff the octant can be covered.

Decidability of (extensions of) $\mathcal{ALC}_{\mathsf{reg}}$ do not transfer well to the non-regular setting.

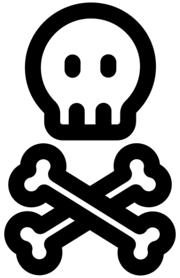
Decidability of (extensions of) $\mathcal{ALC}_{\mathsf{reg}}$ do not transfer well to the non-regular setting.

Loops



Decidability of (extensions of) $\mathcal{ALC}_{\mathsf{reg}}$ do not transfer well to the non-regular setting.

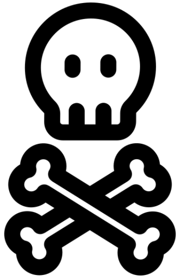
Loops



Nominals

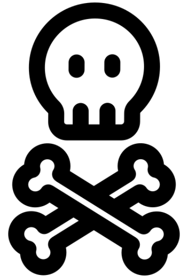
Decidability of (extensions of) $\mathcal{ALC}_{\mathsf{reg}}$ do not transfer well to the non-regular setting.

Loops



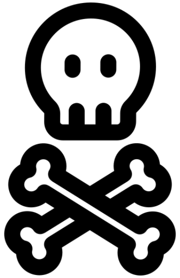
Nominals

Queries



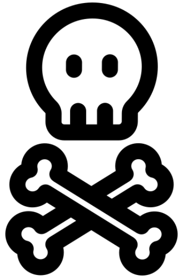
Decidability of (extensions of) $\mathcal{ALC}_{\mathbf{reg}}$ do not transfer well to the non-regular setting.

Loops

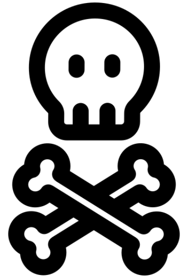


Vis. 1-counter

Nominals

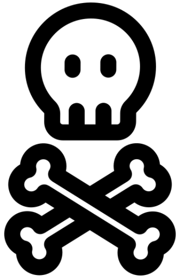


Queries



Decidability of (extensions of) $\mathcal{ALC}_{\mathsf{reg}}$ do not transfer well to the non-regular setting.

Loops

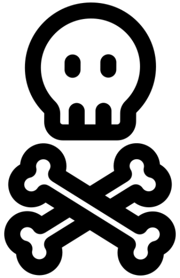


Vis. 1-counter

Nominals

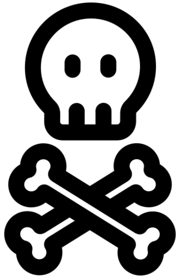
$\mathcal{ALC}_{\mathsf{reg}}^{r\#s\#}$

Queries



Decidability of (extensions of) $\mathcal{ALC}_{\mathsf{reg}}$ do not transfer well to the non-regular setting.

Loops

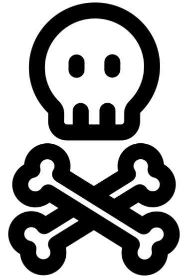


Vis. 1-counter

Nominals

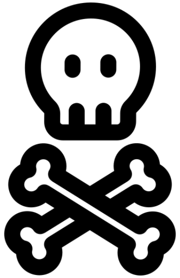
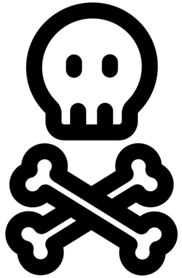
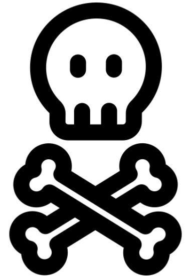
$\mathcal{ALC}_{\mathsf{reg}}^{r\#s\#}$

Queries



$\mathcal{ALC}$ + CRPQs with $r^{\#}s^{\#}$

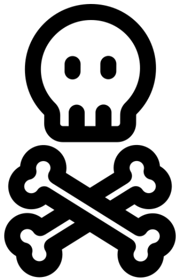
Decidability of (extensions of) $\mathcal{ALC}_{\text{reg}}$ do not transfer well to the non-regular setting.

Loops	**Nominals**	**Queries**
		
Vis. 1-counter	$\mathcal{ALC}_{\text{reg}}^{r^{\#}s^{\#}}$	$\mathcal{ALC}$ + CRPQs with $r^{\#}s^{\#}$

Open Problem 1: Incorporating ABoxes?

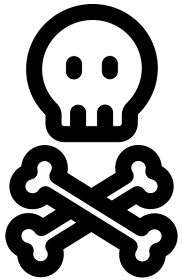
Decidability of (extensions of) $\mathcal{ALC}_{\mathsf{reg}}$ do not transfer well to the non-regular setting.

Loops



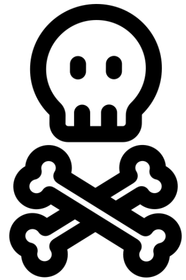
Vis. 1-counter

Nominals



$\mathcal{ALC}_{\mathsf{reg}}^{r^{\#}s^{\#}}$

Queries

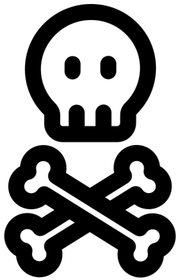
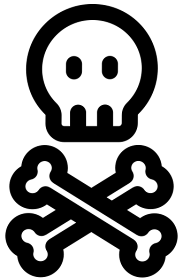
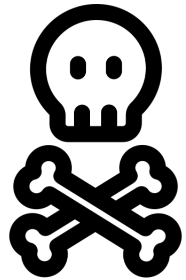


$\mathcal{ALC}$ + CRPQs with $r^{\#}s^{\#}$

Open Problem 1: Incorporating ABoxes?

Open Problem 2: Finite Satisfiability of $\mathcal{ALC}_{\mathsf{vpl}}$?

Decidability of (extensions of) $\mathcal{ALC}_{\mathbf{reg}}$ do not transfer well to the non-regular setting.

Loops	**Nominals**	**Queries**
		
Vis. 1-counter	$\mathcal{ALC}_{\mathrm{reg}}^{r\#s\#}$	$\mathcal{ALC}$ + CRPQs with $r^{\#}s^{\#}$

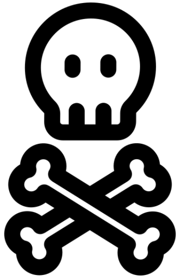
Open Problem 1: Incorporating ABoxes?

Open Problem 2: Finite Satisfiability of $\mathcal{ALC}_{\mathrm{vpl}}$?

Open Problem 3: Sharpen undecidability for $\mathcal{ALC}_{\mathrm{vpl}}$ with Self?

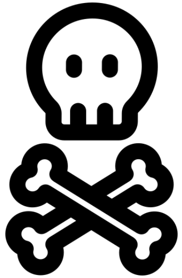
Decidability of (extensions of) $\mathcal{ALC}_{\mathsf{reg}}$ do not transfer well to the non-regular setting.

Loops



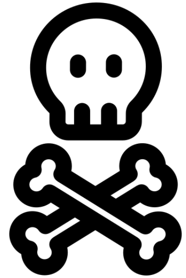
Vis. 1-counter

Nominals



$\mathcal{ALC}^{r^{\#}s^{\#}}_{\mathsf{reg}}$

Queries



$\mathcal{ALC}$ + CRPQs with $r^{\#}s^{\#}$

Looking for (postdoc?) job from Sept'24!

See: bartoszjanbednarczyk.github.io

Open Problem 1: Incorporating ABoxes?

Open Problem 2: Finite Satisfiability of $\mathcal{ALC}_{\mathsf{vpl}}$?

Open Problem 3: Sharpen undecidability for $\mathcal{ALC}_{\mathsf{vpl}}$ with Self?