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

$\mathcal{ALC}_{\text{reg}} := \mathcal{ALC} + \exists \mathcal{L}.C + \forall \mathcal{L}.C$ for all languages $\mathcal{L} \in \text{REG}$. 
Some historical results about $\mathcal{ALC}_{\text{reg}}$ and beyond

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

- Originally developed as a logic for program verification (a.k.a. Propositional Dynamic Logic)
Some historical results about $\mathcal{ALC}_{\text{reg}}$ and beyond

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

- Originally developed as a logic for program verification (a.k.a. Propositional Dynamic Logic)
- $\text{EXPTime}$-complete satisfiability (Pratt 1978).

Can we go beyond regularity?
- CFL ('81)
- $\text{REG} + \#r \#s r \#s + s \#r \#s$
- $\text{REG} + (\text{semi}) \text{ simple minded}$
- More...
Some historical results about $\mathcal{ALC}_{\text{reg}}$ and beyond

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

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

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

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

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

- Originally developed as a logic for program verification (a.k.a. Propositional Dynamic Logic)
- 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 $\mathcal{ALC}_{\text{reg}}$ and beyond

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

- Originally developed as a logic for program verification (a.k.a. Propositional Dynamic Logic)
- \textsc{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 $\mathcal{ALC}_{\text{reg}}$ and beyond

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

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

\begin{itemize}
  \item CFL ('81)
  \item $\text{REG} + \mathcal{r} \# \mathcal{s} \#$
  \item $\text{REG} + \mathcal{r} \# \mathcal{s} \# + \mathcal{s} \# \mathcal{r} \#$
  \item $\text{REG} + (\text{semi}) \text{ simple minded}$
  \item More...
\end{itemize}
Some historical results about $\mathcal{ALC}_{\text{reg}}$ and beyond

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

- Originally developed as a logic for program verification (a.k.a. Propositional Dynamic Logic)
- $\text{ExpTime}$-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?

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

Bartosz “Bart” Bednarczyk

Exploring Non-Regular Extensions of PDL with DL Features
Some historical results about $\mathcal{ALC}_{\text{reg}}$ and beyond

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

- Originally developed as a logic for program verification (a.k.a. Propositional Dynamic Logic)
- $\text{ExpTime}$-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?

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

Bartosz “Bart” Bednarczyk
Exploring Non-Regular Extensions of PDL with DL Features
Some historical results about $\mathcal{ALC}_{\text{reg}}$ and beyond

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

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

Can we go beyond regularity?

- CFL (’81)
- REG $+ r^\# s r^\#$
- REG $+ r^\# s^\# + s^\# r^\#$
Some historical results about $\text{ALC}_{\text{reg}}$ and beyond

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

- Originally developed as a logic for program verification (a.k.a. Propositional Dynamic Logic)
- $\text{ExpTime}$-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?

- CFL ('81)
- $\text{REG} + r^#s^#$
- $\text{REG} + r^#s^# + s^#r^#$
- $\text{REG} + r^#s^#r^#$
- More...

Bartosz “Bart” Bednarczyk

Exploring Non-Regular Extensions of PDL with DL Features
Some historical results about $\mathcal{ALC}_{\text{reg}}$ and beyond

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

- Originally developed as a logic for program verification (a.k.a. Propositional Dynamic Logic)
- $\exp$-Time-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?

- CFL ('81)
- $\mathsf{REG} + r^#s^#$
- $\mathsf{REG} + (\text{semi}) \text{ simple minded}$
- $\mathsf{REG} + r^#s^# + s^#r^#$
Some historical results about $ALC_{reg}$ and beyond

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

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

Can we go beyond regularity?

- $\text{CFL}$ (’81)
- $\text{REG} + r^#s^#$
- $\text{REG} + i^#s^# + s^#r^#$
- $\text{REG} + (\text{semi})$ simple minded
- More...

Bartosz “Bart” Bednarczyk  
Exploring Non-Regular Extensions of PDL with DL Features
**Some historical results about \( \text{ALC}_{\text{reg}} \) and beyond**

\[
\text{ALC}_{\text{reg}} := \text{ALC} + \exists L . C + \forall L . C \quad \text{for all languages } L \in \text{REG}.
\]

- Originally developed as a logic for program verification (a.k.a. Propositional Dynamic Logic)
- \( \text{ExpTime} \)-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?**

- CFL ('81)
- \( \text{REG} + r^#s^# \)
- \( \text{REG} + r^#s^# + s^#r^# \)
- \( \text{REG} + \) (semi) simple minded
- More...

\( \text{ALC} + \text{Visibly Pushdown Languages} \) is decidable.
The success of Visibly Pushdown Languages (VPLs)

\[ \Sigma = \Sigma_c \cup \Sigma_i \cup \Sigma_r \]

- 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 VPL

Why do we care?
- Verification of recursive programs
- XML schema validation

Why not to employ VPL in knowledge representation?
The success of Visibly Pushdown Languages (VPLs)

\[ \Sigma = \Sigma_c \cup \Sigma_i \cup \Sigma_r \text{ (call + internal + returns)} \]

• 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 VPL

Why do we care?
• Verification of recursive programs
• XML schema validation

Why not to employ VPL in knowledge representation?

Bartosz “Bart” Bednarczyk
Exploring Non-Regular Extensions of PDL with DL Features
The success of Visibly Pushdown Languages (VPLs)

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 VPL

Why do we care?
- Verification of recursive programs
- XML schema validation

Why not to employ VPL in knowledge representation?

Bartosz “Bart” Bednarczyk
Exploring Non-Regular Extensions of PDL with DL Features
The success of Visibly Pushdown Languages (VPLs)

\[ \text{VPL} = \text{Finite Automata} + \text{Input-Driven Stack} \]

\[ \Sigma = \Sigma_c \cup \Sigma_i \cup \Sigma_r \text{(call + internal + returns)} \]

- 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 VPL

Why do we care?
- Verification of recursive programs
- XML schema validation

Why not to employ VPL in knowledge representation?
The success of Visibly Pushdown Languages (VPLs)

\[ \text{VPL = Finite Automata + Input-Driven Stack} \]

\[ \Sigma = \Sigma_c \cup \Sigma_i \cup \Sigma_r \text{ (call + internal + returns)} \]
The success of Visibly Pushdown Languages (VPLs)

\[ \text{VPL} = \text{Finite Automata} + \text{Input-Driven Stack} \]

\[ \Sigma = \Sigma_c \cup \Sigma_i \cup \Sigma_r \text{ (call + internal + returns)} \]

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 VPL

Why do we care?
- Verification of recursive programs
- XML schema validation

Why not to employ VPL in knowledge representation?
The success of Visibly Pushdown Languages (VPLs)

\[ \text{VPL} = \text{Finite Automata} + \text{Input-Driven Stack} \]

\[ \Sigma = \Sigma_c \cup \Sigma_i \cup \Sigma_r \text{ (call + internal + returns)} \]

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 VPL

Why do we care?

- Verification of recursive programs
- XML schema validation

Why not to employ VPL in knowledge representation?

Bartosz “Bart” Bednarczyk  Exploring Non-Regular Extensions of PDL with DL Features  2 / 8
The success of Visibly Pushdown Languages (VPLs)

VPL = Finite Automata + Input-Driven Stack

\[ \Sigma = \Sigma_c \cup \Sigma_i \cup \Sigma_r \text{ (call + internal + returns)} \]

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)

\[ \text{VPL} = \text{Finite Automata} + \text{Input-Driven Stack} \]

\[ \Sigma = \Sigma_c \cup \Sigma_i \cup \Sigma_r \text{ (call + internal + returns)} \]

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 \( \text{VPL} \)

Why do we care?
- Verification of recursive programs
- XML schema validation

Why not to employ \( \text{VPL} \) in knowledge representation?
The success of Visibly Pushdown Languages (VPLs)

\[
\text{VPL} = \text{Finite Automata} + \text{Input-Driven Stack}
\]

\[
\Sigma = \Sigma_c \cup \Sigma_i \cup \Sigma_r \text{ (call + internal + returns)}
\]

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 \text{VPL}

Why do we care?
- Verification of recursive programs
- XML schema validation

Why not to employ \text{VPL} in knowledge representation?
The success of Visibly Pushdown Languages (VPLs)

\[ \text{VPL} = \text{Finite Automata + Input-Driven Stack} \]

\[ \Sigma = \Sigma_c \cup \Sigma_i \cup \Sigma_r \text{ (call + internal + returns)} \]

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 \( \text{VPL} \)

Why do we care?

<table>
<thead>
<tr>
<th>Decision problems for automata</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Emptiness</td>
<td>Universality/Equivalence</td>
</tr>
<tr>
<td>NFA</td>
<td>\text{NLOGSPACE}</td>
</tr>
<tr>
<td>PDA</td>
<td>\text{PTIME}</td>
</tr>
<tr>
<td>DPDA</td>
<td>\text{PTIME}</td>
</tr>
<tr>
<td>VPA</td>
<td>\text{PTIME}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Closure under</th>
<th>Union</th>
<th>Intersection</th>
<th>Complement</th>
<th>Concat.</th>
<th>Kleene-*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>CFL</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DCFL</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>VPL</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Bartosz “Bart” Bednarczyk
Exploring Non-Regular Extensions of PDL with DL Features
The success of Visibly Pushdown Languages (VPLs)

VPL = Finite Automata + Input-Driven Stack

\[ \Sigma = \Sigma_c \cup \Sigma_i \cup \Sigma_r \text{ (call + internal + returns)} \]

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 VPL

Why do we care?

- Verification of recursive programs

<table>
<thead>
<tr>
<th>Decision problems for automata</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Emptiness</td>
<td>Universality/Equivalence</td>
</tr>
<tr>
<td>NFA</td>
<td>NLOGSPACE</td>
</tr>
<tr>
<td>PDA</td>
<td>PTIME</td>
</tr>
<tr>
<td>DPDA</td>
<td>PTIME</td>
</tr>
<tr>
<td>VPA</td>
<td>PTIME</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Closure under</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Union</td>
<td>Intersection</td>
</tr>
<tr>
<td>Regular</td>
<td>Yes</td>
</tr>
<tr>
<td>CFL</td>
<td>Yes</td>
</tr>
<tr>
<td>DCFL</td>
<td>No</td>
</tr>
<tr>
<td>VPL</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Bartosz “Bart” Bednarczyk  Exploring Non-Regular Extensions of PDL with DL Features  2 / 8
The success of Visibly Pushdown Languages (VPLs)

\[ \text{VPL} = \text{Finite Automata} + \text{Input-Driven Stack} \]

\[ \Sigma = \Sigma_c \cup \Sigma_i \cup \Sigma_r \text{ (call + internal + returns)} \]

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 VPL

Why do we care?

- Verification of recursive programs
- XML schema validation

<table>
<thead>
<tr>
<th>Decision problems for automata</th>
<th>Emptyness</th>
<th>Universality/Equivalence</th>
<th>Inclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>NFA</td>
<td>NLOGSPACE</td>
<td>PSPACE</td>
<td>PSPACE</td>
</tr>
<tr>
<td>PDA</td>
<td>PTIME</td>
<td>Undecidable</td>
<td>Undecidable</td>
</tr>
<tr>
<td>DPDA</td>
<td>PTIME</td>
<td>Decidable</td>
<td>Undecidable</td>
</tr>
<tr>
<td>VPA</td>
<td>PTIME</td>
<td>EXPTIME</td>
<td>EXPTIME</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Closure under</th>
<th>Union</th>
<th>Intersection</th>
<th>Complement</th>
<th>Concat.</th>
<th>Kleene-*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>CFL</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DCFL</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>VPL</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Bartosz “Bart” Bednarczyk
Exploring Non-Regular Extensions of PDL with DL Features
The success of Visibly Pushdown Languages (VPLs)

\[ \text{VPL} = \text{Finite Automata} + \text{Input-Driven Stack} \]

\[ \Sigma = \Sigma_c \cup \Sigma_i \cup \Sigma_r \text{ (call + internal + returns)} \]

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 VPL

Why do we care?

- Verification of recursive programs
- XML schema validation

Why not to employ VPL in knowledge representation?

Bartosz “Bart” Bednarczyk
Exploring Non-Regular Extensions of PDL with DL Features
Is $\mathcal{ALC}_{vpl}$ robust under extensions with features supported by W3C ontology languages?

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

• $\mathcal{ALC}_{vpl}$ is inverses is undecidable (unpublished, discovered in Stefan Göller's PhD Thesis'2008)

How about other features? How about querying?

Loops
Nominals
Queries

Visibly one counter $\mathcal{ALC}_S$ TBoxes + CRPQs with $r#s#reg$
Is $ALC_{vpl}$ robust under extensions with features supported by W3C ontology languages?

- $ALC_{vpl}$ is decidable and $2\text{ExpTime}$-complete (Löding et. al 2007)

How about other features? How about querying?

Loops
Nominals
Queries

$ALC_r$ # $s$ # $reg$

Bartosz “Bart” Bednarczyk
Exploring Non-Regular Extensions of PDL with DL Features
Is $\mathcal{ALC}_{vpl}$ robust under extensions with features supported by W3C ontology languages?

- $\mathcal{ALC}_{vpl}$ is decidable and 2ExpTime-complete (Löding et. al 2007)
- $\mathcal{ALC}_{vpl}$ is inverses is undecidable (unpublished, discovered in Stefan Göller’s PhD Thesis’2008)
Is $\mathcal{ALC}_{vpl}$ robust under extensions with features supported by W3C ontology languages?

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

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

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

How about other features? How about querying?

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

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

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

How about other features? How about querying?

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

Bartosz Bednarczyk

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

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

How about other features? How about querying?

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

Bartosz Bednarczyk$^{1,2}$
Is $\mathcal{ALC}_{vpl}$ robust under extensions with features supported by W3C ontology languages?

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

How about other features? How about querying?

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

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

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

How about other features? How about querying?

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

Bartosz Bednarczyk

Loops  Nominals  Queries

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

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

How about other features? How about querying?

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

Bartosz Bednarczyk

Loops

Visibly one counter

Nominals

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

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

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

How about other features? How about querying?

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

Bartosz Bednarczyk$^{1,2}$
Proof sketch: Undecidability of $\mathcal{ALC}_{\text{vpl}} + \text{Self}$

Input: Deterministic one counter automata $A_1, A_2$.
Output: Is $L(A_1) \cap L(A_2)$ non-empty?

Valiant 1973

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

Given DOCA $A_1, A_2$, we get VPA $\hat{A}_1, \hat{A}_2$ projectively recognizing their lang. + $\hat{C}_1, \hat{C}_2$ for complements.

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

Trick 2: Employ concepts $\forall \hat{A}_1 . \text{OK}_1 \sqcap \forall \hat{C}_1 . \neg \text{OK}_1$ to decorate interpretations with “acceptance” of $A_1$.
Proof sketch: Undecidability of $\mathcal{ALC}_{vpl} + \text{Self}$

**Input**: Deterministic one counter automata $A_1, A_2$. 

Valiant 1973

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

Given DOCA $A_1, A_2$, we get VPA $\hat{A}_1, \hat{A}_2$ projectively recognizing their lang. $\hat{C}_1, \hat{C}_2$ for complements.

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

Trick 2: Employ concepts $\forall \hat{A}_1, \text{OK}_1 \cap \forall \hat{C}_1, \neg \text{OK}_1$ to decorate interpretations with "acceptance" of $A_1$. 

Bartosz “Bart” Bednarczyk  Exploring Non-Regular Extensions of PDL with DL Features
Proof sketch: Undecidability of $\mathcal{ALC}_{vpl} + \text{Self}$

**Input:** Deterministic one counter automata $A_1, A_2$.

**Output:** Is $L(A_1) \cap L(A_2)$ non-empty?

Valiant 1973

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

Given DOCA $A_1, A_2$, we get VPA $\hat{A}_1, \hat{A}_2$ projectively recognizing their lang. + $\hat{C}_1, \hat{C}_2$ for complements.

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

Trick 2: Employ concepts $\forall \hat{A}_1, \forall \hat{C}_1, \neg OK_1$ to decorate interpretations with "acceptance" of $A_1$. 

Bartosz “Bart” Bednarczyk  Exploring Non-Regular Extensions of PDL with DL Features
Proof sketch: Undecidability of $\mathcal{ALC}_{vpl} + \text{Self}$

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

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

Valiant 1973
Proof sketch: Undecidability of $\mathcal{ALC}_{vpl} + \text{Self}$

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

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

Key insight: Deterministic one-counter languages can be projectively recognized by VPA.
Proof sketch: Undecidability of $\mathcal{ALC}_{vpl} + \text{Self}$

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

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

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

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

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

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

**Valiant 1973**

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

**Lemma 15.** For any DOCA $\mathcal{A}$ over $\Sigma$, we can construct a VOCA $\hat{\mathcal{A}}$ over

$\hat{\Sigma} := (\Sigma \times \{c\}, (\Sigma \times \{i\}) \cup \{x\}, \Sigma \times \{r\})$ where $x$ is a fresh internal letter, such that all words in $L(\mathcal{A})$ have the form $\tilde{a}_1x\tilde{a}_2x\ldots x\tilde{a}_n$ for $\tilde{a}_1, \ldots, \tilde{a}_n \in \Sigma \times \{c, i, r\}$, and $L(\hat{\mathcal{A}}) = \{ \pi_1(\tilde{w}) \mid \tilde{w} := \tilde{a}_1 \ldots \tilde{a}_n, \quad \tilde{a}_1x\ldots x\tilde{a}_n \in L(\hat{\mathcal{A}}) \}$ holds.

Given DOCA $\mathcal{A}_1, \mathcal{A}_2$, we get VPA $\hat{\mathcal{A}}_1, \hat{\mathcal{A}}_2$ projectively recognizing their lang. + $\hat{C}_1, \hat{C}_2$ for complements.
Proof sketch: Undecidability of $ALC_{vpl} + \text{Self}$

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

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

Valiant 1973

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

**Lemma 15.** For any DOCA $\mathcal{A}$ over $\Sigma$, we can construct a VOCA $\hat{\mathcal{A}}$ over
\[
\hat{\Sigma} := (\Sigma \times \{c\}, (\Sigma \times \{i\}) \cup \{x\}, \Sigma \times \{r\})
\]
where $x$ is a fresh internal letter, such that all words in $L(\mathcal{A})$ have the form $\tilde{a}_1x\tilde{a}_2x\ldots x\tilde{a}_n$ for $\tilde{a}_1, \ldots, \tilde{a}_n \in \Sigma \times \{c, i, r\}$, and $L(\hat{\mathcal{A}}) = \{\pi_1(\tilde{w}) \mid \tilde{w} := \tilde{a}_1 \ldots \tilde{a}_n, \ \tilde{a}_1x\ldots x\tilde{a}_n \in L(\hat{\mathcal{A}})\}$ holds.

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

**Trick 1:** Encode “word-like structures” with loops storing the actual letters.
Proof sketch: Undecidability of $\mathcal{ALC}_{vpl} + \text{Self}$

**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 $\Sigma$, we can construct a VOCA $\tilde{\mathcal{A}}$ over 
$
\tilde{\Sigma} := (\Sigma \times \{c\}, (\Sigma \times \{i\}) \cup \{x\}, \Sigma \times \{r\})
$
where $x$ is a fresh internal letter, such that all words in $\mathcal{L}(\mathcal{A})$ have the form $\tilde{a}_1x\tilde{a}_2x\ldots x\tilde{a}_n$ for $\tilde{a}_1, \ldots, \tilde{a}_n \in \Sigma \times \{c, i, r\}$, and $\mathcal{L}(\mathcal{A}) = \{\pi_1(\tilde{w}) \mid \tilde{w} := \tilde{a}_1 \ldots \tilde{a}_n, \ \tilde{a}_1x\ldots x\tilde{a}_n \in \mathcal{L}(\tilde{\mathcal{A}}) \}$ holds.

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

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

(a, c), (a, r)  (b, c), (b, r)  (b, c), (b, r)  (a, c), (a, r)  (c, c), (c, r)

(a, i)  (b, i)  (b, i)  (a, i)  (c, i)
Proof sketch: Undecidability of $\mathcal{ALC}_{vpl} + \text{Self}$

**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 $\Sigma$, we can construct a VOCA $\hat{\mathcal{A}}$ over

$$\hat{\Sigma} := (\Sigma \times \{c\}, (\Sigma \times \{i\}) \cup \{x\}, \Sigma \times \{r\})$$

where $x$ is a fresh internal letter, such that all words in $\mathcal{L}(\mathcal{A})$ have the form $\tilde{a}_1 x \tilde{a}_2 x \ldots x \tilde{a}_n$ for $\tilde{a}_1, \ldots, \tilde{a}_n \in \Sigma \times \{c, i, r\}$, and $\mathcal{L}(\mathcal{A}) = \{\pi_1(\tilde{w}) \mid \tilde{w} := \tilde{a}_1 \ldots \tilde{a}_n, \tilde{a}_1 x \ldots x \tilde{a}_n \in \mathcal{L}(\hat{\mathcal{A}})\}$ 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: $\text{abbac}$

**Trick 2:** Employ concepts $\forall \hat{\mathcal{A}}_1. \text{OK}_1 \cap \forall \hat{\mathcal{C}}_1. \neg \text{OK}_1$ to decorate interpretations with “acceptance” of $\mathcal{A}_1$.  

Bartosz “Bart” Bednarczyk  
Exploring Non-Regular Extensions of PDL with DL Features
Proof sketch: Undecidability of $\mathcal{ALCO} + r\#s\#$ (Introduction)

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

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

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

**Input:** A finite set of 4-sided tiles with a distinguished colour □.
Proof sketch: Undecidability of $ALCO + r\#s\#$ (Introduction)

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

**Output:** Is there $N, M \in \mathbb{N}$ so that we can cover a □-bordered $(N \times M)$ 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 □.

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

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

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

Bartosz “Bart” Bednarczyk  Exploring Non-Regular Extensions of PDL with DL Features
Proof sketch: Undecidability of $ALCO + r\#s\#$ (Introduction)

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

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

Fig. 1: If $Col = \{\square, \Box, \text{col}1, \text{col}2\}$ and $T = Col^4$, the map $\xi := \{(0, 0) \mapsto \Box, (1, 0) \mapsto \Box, (2, 0) \mapsto \Box, (3, 0) \mapsto \Box, (0, 1) \mapsto \Box, (1, 1) \mapsto \Box, (2, 1) \mapsto \Box, (3, 1) \mapsto \Box, (0, 2) \mapsto \Box, (1, 2) \mapsto \Box, (2, 2) \mapsto \Box, (3, 2) \mapsto \Box\}$ 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 $ALCO + r^s$ (Introduction)

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

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

![Visualization of $\xi$.](image1)

![The encoding of $\xi$ as a $D$-snake $\mathcal{I}$.](image2)

**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?

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

Bartosz “Bart” Bednarczyk  
Exploring Non-Regular Extensions of PDL with DL Features
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 $st \rightarrow md$ and $md \rightarrow end$ are all equal to $N$.

We synchronize 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 reach $end$. 

Bartosz “Bart” Bednarczyk
Exploring Non-Regular Extensions of PDL with DL Features
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 $st \leadsto md$ and $md \leadsto end_t$ 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 $st \leadsto md$ and $md \leadsto end_t$ are all equal to $N$.

We synchronize 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 \( \text{st} \rightsquigarrow \text{md} \) and \( \text{md} \rightsquigarrow \text{end}_t \) are all equal to \( N \).

We synchronize 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 $st \leadsto md$ and $md \leadsto end_t$ are all equal to $N$.

We synchronize 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 $end_t$.
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.

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

**Key Property:** $C \not| = q$ iff the octant can be covered.
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 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?

### Key Property:

$C \not|= q \iff$ the octant can be covered.
**Proof sketch: Undecidability of querying $\text{ALC}$-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.
Proof sketch: Undecidability of querying $\text{ALC}$-TBoxes with non-regular queries

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

**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.

\[
q^D := \bigvee_{t, t' \text{ violating (OHori)}} \left[ r(x_1, x_2) \land r^*(x_2, y_1) \land r(y_1, y_2) \land s^*(y_1, z_1) \land s^*(y_2, z_2) \land \tau^# s^#(x_1, z_1) \land \tau^# s^#(x_2, z_2) \land C_t(z_1) \land C_{t'}(z_2) \right]
\]
Proof sketch: Undecidability of querying $\mathcal{ALC}$-TBoxes with non-regular queries

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

**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.

**Key Property:** $C \not\models q$ iff the octant can be covered.
Decidability of (extensions of) $\mathcal{ALC}_{reg}$ do not transfer well to the non-regular setting.
Decidability of (extensions of) $\text{ALC}_{\text{reg}}$ do not transfer well to the non-regular setting.

**Loops**

![Skull and Crossbones](image)
Decidability of (extensions of) $\mathcal{ALC}_{\text{reg}}$ do not transfer well to the non-regular setting.

**Open Problem 1:** Incorporating ABoxes?

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

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

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

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

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

Loops  Nominals  Queries

Vis. 1-counter
Decidability of (extensions of) $\mathcal{ALC}_{\text{reg}}$ do not transfer well to the non-regular setting.

Loops | Nominals | Queries
---|---|---
[Image of skull and crossbones] | [Image of skull and crossbones] | [Image of skull and crossbones]
Vis. 1-counter | $\mathcal{ALC}_{\text{reg}}^{r\#s\#}$ |
Decidability of (extensions of) $\mathcal{ALC}_{\text{reg}}$ do not transfer well to the non-regular setting.

**Loops**
- Vis. 1-counter

**Nominals**
- $\mathcal{ALC}_{\text{reg}}^{r\#s\#}$

**Queries**
- $\mathcal{ALC}$ + CRPQs with $r\#s\#$

Open Problem 1: Incorporating ABoxes?
Open Problem 2: Finite Satisfiability of $\mathcal{ALC}_{\text{vpl}}$?
Open Problem 3: Sharpen undecidability for $\mathcal{ALC}_{\text{vpl}}$ with Self?

Looking for (postdoc?) job from Sept'24!
See: bartoszjanbednarczyk.github.io
Decidability of (extensions of) $\mathcal{ALC}_{\text{reg}}$ do not transfer well to the non-regular setting.

**Loops**
- Vis. 1-counter

**Nominals**
- $\mathcal{ALC}_{\text{reg}}$

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

Open Problem 1: Incorporating ABoxes?
Decidability of (extensions of) $\mathcal{ALC}_{\text{reg}}$ do not transfer well to the non-regular setting.

<table>
<thead>
<tr>
<th>Loops</th>
<th>Nominals</th>
<th>Queries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vis. 1-counter</td>
<td>$\mathcal{ALC}_{\text{reg}}^{r#s#}$</td>
<td>$\mathcal{ALC} + \text{CRPQs with } r#s#$</td>
</tr>
</tbody>
</table>

Open Problem 1: Incorporating ABoxes?
Open Problem 2: Finite Satisfiability of $\mathcal{ALC}_{\text{vpl}}$?
Decidability of (extensions of) $\mathcal{ALC}_{\text{reg}}$ do not transfer well to the non-regular setting.

Open Problem 1: Incorporating ABoxes?

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

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

Vis. 1-counter, $\mathcal{ALC}_{\text{reg}}^{r#s#}$, $\mathcal{ALC}$ + CRPQs with $r#s#$
Decidability of (extensions of) $\mathcal{ALC}_{\text{reg}}$ do not transfer well to the non-regular setting.

<table>
<thead>
<tr>
<th>Loops</th>
<th>Nominals</th>
<th>Queries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vis. 1-counter</td>
<td>$\mathcal{ALC}_{\text{reg}}^r s#$</td>
<td>$\mathcal{ALC}$ + CRPQs with $r s#$</td>
</tr>
</tbody>
</table>

Open Problem 1: Incorporating ABoxes?
Open Problem 2: Finite Satisfiability of $\mathcal{ALC}_{\text{vpl}}$?
Open Problem 3: Sharpen undecidability for $\mathcal{ALC}_{\text{vpl}}$ with Self?

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

See: bartoszjanbednarczyk.github.io