Our application: **situation recognition for context-aware systems**

- observe complex hard- & software system
  Goal: save energy

- recognize situations, where adaptation is beneficial

- **Description Logic-based** approach:
  TBox for background knowledge, ABox for observations

- employ Description Logic reasoning: **conjunctive query answering**

- collecting data from several sources! — **sensors**
• data (& ontology) stored in a data base

• sliding windows approach to monitored data
  - observations are recorded at fixed intervals
  - sequence of system snapshots is obtained

• raw data is preprocessed by data base
  - raw data is ‘cleaned’ and aggregated

• situation descriptions are stored as DB queries
Why fuzzy information?

Represent sensor values ‘faithfully’

- Logic: symbolic representation of values
- values: low, medium, high
- small change in value $\rightarrow$ similar classification
- can level out noise
Description logics and fuzzy information

Semantics:

- Classical logic: \( \{0, 1\} \) binary
- Fuzzy logic: \( \{0, \cdots, 1\} \) infinitely many membership degrees

Results on DL & fuzzy information:

- easily undecidable!
- decidable, if finitely many membership degrees are used.
  But: costly reduction!

Our goal: employ reasoner for DL-Lite\(_R\) (a.k.a. OWL 2 QL)

- DL-Lite\(_R\)-reasoners: optimized implementations!
- only the ABox and the queries are fuzzy!
  (TBox stays crisp)
The description logic $\text{DL-Lite}_R$ and its fuzzy variant

**DL-Lite}_R concepts:**

\[
B \rightarrow A \mid \exists Q \quad C \rightarrow \top \mid B \mid \neg B \\
Q \rightarrow P \mid P^- \quad R \rightarrow Q \mid \neg Q
\]

**DL-Lite}_R axioms:**

\[
B \sqsubseteq C \quad Q \sqsubseteq R \quad \text{funct}(Q)
\]

**Fuzzy DL-Lite}_R assertions:**

\[
\langle B(a), d \rangle \quad \langle P(a, b), d \rangle
\]

**Interpretation for fuzzy DL-Lite}_R:**

\[
\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})
\]

- **individuals:**
  \[
a^\mathcal{I} \in \Delta^\mathcal{I}
\]

- **named concepts:**
  \[
  A^\mathcal{I} : \Delta^\mathcal{I} \rightarrow [0, 1]
  \]

- **atomic roles:**
  \[
  P^\mathcal{I} : \Delta^\mathcal{I} \times \Delta^\mathcal{I} \rightarrow [0, 1]
  \]
Fuzzy operators and semantics of complex concepts

<table>
<thead>
<tr>
<th>t-norm $a \otimes b$</th>
<th>negation $\ominus a$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gödel</td>
<td>$\min(a, b)$</td>
</tr>
<tr>
<td>Lukasiewicz</td>
<td>$\max(a + b - 1, 0)$</td>
</tr>
<tr>
<td>Product</td>
<td>$a \times b$</td>
</tr>
</tbody>
</table>

\[ \begin{cases} 1, & a = 0 \\ 0, & a > 0 \end{cases} \]

\[ 1 - a \]

\[ \begin{cases} 1, & a = 0 \\ 0, & a > 0 \end{cases} \]

Let: $\delta, \delta' \in \Delta^I$

**complex concepts:**

- $(\exists Q)^I(\delta) = \sup_{\delta' \in \Delta^I} Q^I(\delta, \delta')$
- $(\neg B)^I(\delta) = \ominus B^I(\delta)$
- $\top^I(\delta) = 1$

**fuzzy conjunctive query:**

\[ q(\vec{x}) = \exists \vec{y}. \varphi(\vec{x}, \vec{y}) \geq d \]
\[ \mathcal{T} := \{ \text{Server} \sqsubseteq \exists \text{hasCPU}, \ \exists \text{hasCPU}^\bot \sqsubseteq \text{CPU}, \ \text{func(\text{hasCPU}^\bot)} \} \]

\[ \mathcal{A} := \{ \langle \text{Server(server}_1), 1 \rangle, \langle \text{hasCPU(server}_1, \text{cpu}_1), 1 \rangle, \langle \text{OverUtilized}(...) \rangle \} \]

\[ q_1(x) = \text{CPU}(x) \]
\[ q_2(x, y) = \text{hasCPU}(x, y) \land \text{OverUtilized}(y) \]
\[ q_3(x) = \exists y \ \text{hasCPU}(x, y) \land \text{OverUtilized}(y) \]

\[ \text{ans}(q_1(x), \varnothing) = \{(\text{cpu}_1, 1), (\text{cpu}_2, 1)\} \]
\[ \text{ans}(q_2(x, y), \varnothing) = \{(\text{server}_1, \text{cpu}_1, 0.6), (\text{server}_2, \text{cpu}_2, 0.8)\} \]
\[ \text{ans}(q_3(x), \varnothing) = \{(\text{server}_1, 0.8)\} \].
1.) Use TBox to reformulate query $q$ into a FOL-query $q_T$, discard the TBox

"Compile TBox information into the query."

2.) View ABox $\mathcal{A}$ as a relational database $\mathcal{I}_\mathcal{A}$, where

- $A^{\mathcal{I}_\mathcal{A}} = \{a \mid A(a) \in \mathcal{A}\}$
- $r^{\mathcal{I}_\mathcal{A}} = \{(a, b) \mid r(a, b) \in \mathcal{A}\}$

3.) Evaluate $q_T$ in the DB $\mathcal{I}_\mathcal{A}$ using a relational query engine.
The black box rewriting algorithm for fuzzy queries

1. Compute **crisp query rewriting** \( q(\bar{x}) \sim q_T(\bar{x}) \)

2. **rewrite** query \( q_T(\bar{x}) \) by

   - **degree variables**: placeholders for degrees, e.g.: \( x_d \)
   - **replacing**
     - concepts in concept atoms by binary predicates
     - roles in role atoms by ternary predicates
   
   where new argument is a degree variable
   
   (to prepare for the tables storing fuzzy information)

   - **degree atoms**:
     - to capture the fuzzy operators, e.g.: \( \Phi_\times(x_d, x_d'), \Phi_\ominus(x_d) \)
     - to ensure conditions on degrees, e.g.: \( \Phi_\geq(x_d, 0.3) \)
1. Compute crisp query rewriting \( q(\vec{x}) \rightsquigarrow q_T(\vec{x}) \)

2. **rewrite** query \( q_T(\vec{x}) \rightsquigarrow q_{T,f}(\vec{x}, \vec{x_d}) \)

3. **Evaluate** rewritten query \( q_{T,f}(\vec{x}, \vec{x_d}) \) over the ‘fuzzy’ database

4. Keep **highest degree** for each returned tuple
FLite Reasoner

Query rewriting by Ontop

Query rewriting to treat fuzzy degrees

Query evaluation by SQL query engine

Answers to $\langle \varphi, d \rangle$

Fuzzy ABox

Context TBox

Fuzzy Query $\langle q(x), d \rangle$
The FLite reasoner

Our implementation: FLite

- implements the blackbox rewriting approach for Gödel semantics
- employs Ontop to compute the crisp rewriting of the query
- stores fuzzy ABox in database

Empirical test:

- **Test ontology**: TBox with 311 axioms (178 concepts, 39 roles)
  ABox (/database) with 10 tables (4 fuzzy ones)

- **Test queries**: fuzzy conjunctive queries
  (with 13 atoms, 9 fuzzy ones \(\sim\) 9 degree variables)

- **Test set-up**: measure running times of FLite and of Ontop
Performance measurement

Query answering time (ms)

Number of ABox database assertions

FLite reasoner
Ontop reasoner
Temporal fuzzy queries over fuzzy DL-Lite ontologies

Recall:

- temporal operators in the TBox cause undecidability
- temporal information given by sequence of (fuzzy) ABoxes
- navigate “backwards” on the sequence

Temporal queries use the operators from LTL:

- $\square^-$ ‘always in the past’
- $\diamond^-$ ‘eventually in the past’
- $\circ^-$ ‘previous’

Temporal query: query “wrapped” in LTL operators $\circ^- \square^- \langle q(\vec{x}), d \rangle$
Fuzzy ABox

Context TBox

Fuzzy Query

Temporal fuzzy Reasoner

Query rewriting by Ontop

Query rewriting to treat fuzzy degrees

Query rewriting to treat LTL operators

Query evaluation by SQL query engine

Answers to

$\langle q(\bar{x}), d \rangle$
Conclusions

- introduced a decidable variant of temporal fuzzy query answering

- proposed an pragmatic approach to temporal fuzzy query answering that enriches crisp query rewritings for DL-Lite$_R$

- method implemented on top of OnTop reasoner

- first evaluation suggest: acceptable overhead for fuzzy information Temporal fuzzy QA?