Default Reasoning on Top of Ontologies with dl-Programs

DAO Tran Minh
Supervisor: Prof. Thomas Eiter

Vienna University of Technology

June 12, 2008
## Outline

1. **Motivating Example**
2. **Introduction**
   - Default Logic at a glance
   - An overview of dl-programs
   - From Default Logic to dl-programs
3. **Transformations**
   - Some conventions
   - Select-defaults-and-check based transformations
   - Select-justifications-and-check based transformation
4. **Experimental Results**
   - Transformation Π
   - Transformation Ω
   - Transformation Υ
   - Compare 3 transformations in caching mode
5. **Conclusions and Future work**
Simple bird ontology
- \( Flier \sqsubseteq \neg \text{NonFlier} \)
- \( Penguin \sqsubseteq \text{Bird} \)
- \( Penguin \sqsubseteq \text{NonFlier} \)
- \( Penguin(\text{tweety}) \)
- \( Bird(\text{joe}) \)

How to enable default reasoning on top of ontologies?
First attempt to embed default reasoning into terminological knowledge representation by Baader (1993)
Integration of rules and ontologies
One of the most famous nonmonotonic reasoning formalizations.

Default rules: \[ \frac{\alpha(X) : \beta_1(X), ..., \beta_m(X)}{\gamma(X)} \]

Default theory: \[ T = \langle W, D \rangle \].

The totality of knowledge induced by a default theory: extension.

Our purpose: allow each \( \alpha, \beta, \gamma \) to be either a concept or a role name in a DL-KB. For instance:

\[
\begin{align*}
Bird(X) : Flier(X) \\
\Rightarrow Flier(X)
\end{align*}
\]
An overview of dl-programs

- **Theoretical point of view:**
  - an approach on the integration of rules and ontologies
  - key idea: DL atoms which allow us to update and query the DL-KB
  - Eg:
    \[ DL\left[\text{WhiteWine} \sqcup \text{iswhitewhine}; \neg \text{WhiteWine}\right](X). \]
    
    - input list for updating
    - query

- strict semantics integration

- **Practical point of view:**
  - dlvhex: a prover for Semantics Web Reasoning under Answer-Set Semantics, available with a plugin environment
  - dlvhex-dlplugin: allows the use of DL atoms, communicates with a DL-KB via RacerPro
From Default Logic to dl-programs

Motivating Example

Introduction

Transformations

Experimental Results

Conclusions and Future work

User

default rules

[A(X);B(X)]

/[C(X)]

ontology

dl-rules

HEX-rules

from ontology

dfconverter

dl-rules

C(X):-DL[\lambda;A](X),

not DL[\lambda;-B](X).

dl-rules

HEX-rules

Models/Extensions

dlconverter

dlvhex

dl-rules

ontology

HEX-rules

DAO Tran Minh

Vienna University of Technology

Default Reasoning on Top of Ontologies with dl-Programs
Some conventions

- Default theory $\Delta = \langle L, D \rangle$; $L$ is a DL knowledge base, $D \equiv \{\delta_1, \ldots, \delta_n\}$
- $\delta \equiv \frac{\alpha(\vec{X}) : \beta_1(\vec{Y}_1), \ldots, \beta_m(\vec{Y}_m)}{\gamma(\vec{Z})}$
- $\text{name}(\gamma)$: predicate name of the literal $\gamma$
- $\text{aux}_\gamma$:
  - $\text{in}_\text{name}(\gamma)$ if $\gamma$ is positive
  - $\text{in}_\text{not}_\text{name}(\gamma)$ if $\gamma$ is negative
- $\text{auxc}_\beta_i$:
  - $\text{cons}_\text{name}(\beta_i)$ if $\beta_i$ is positive
  - $\text{cons}_\text{not}_\text{name}(\beta_i)$ if $\beta_i$ is negative
Select-defaults-and-check based transformations

Transformation Π

- Rules that guess whether δ’s conclusion belongs to the extension E:
  \[ aux_\gamma(\overrightarrow{Z}) \leftarrow \text{not } out_aux_\gamma(\overrightarrow{Z}). \]
  \[ out_aux_\gamma(\overrightarrow{Z}) \leftarrow \text{not } aux_\gamma(\overrightarrow{Z}). \]

- A rule that checks the compliance of the guess for E with L
  \[ \text{fail} \leftarrow DL[\lambda'; \gamma](\overrightarrow{Z}), out_aux_\gamma_i(\overrightarrow{Z}), \text{not fail}. \]
  where \( \lambda' \equiv \bigcup_{\delta_i \in D} (\gamma_i \cup \text{in}_\text{name}(\gamma_i); \gamma_i \cup \text{in}_\text{not}_\text{name}(\gamma_i)) \)

- A rule for applying δ as in \( \Gamma_{\Delta}(E) \)
  \[ p_aux_\gamma(\overrightarrow{Z}) \leftarrow DL[\lambda; \alpha](\overrightarrow{X}), \]
  \[ \text{not } DL[\lambda; \neg \beta_1](\overrightarrow{Y_1}), \ldots, \text{not } DL[\lambda; \neg \beta_m](\overrightarrow{Y_m}). \]
  where \( \lambda \equiv \bigcup_{\delta_i \in D} (\gamma_i \cup p_in_name(\gamma_i); \gamma_i \cup p_in_not_name(\gamma_i)) \)
Transformation Π - cont.

- Rules which check whether $E$ and $\Gamma_\Delta(E)$ coincide:

  $\text{fail} \leftarrow \text{not } DL[\lambda; \gamma](\vec{Z}), \text{aux}_\gamma(\vec{Z}), \text{not fail}$.

  $\text{fail} \leftarrow DL[\lambda; \gamma](\vec{Z}), \text{out}_\gamma(\vec{Z}), \text{not fail}$.

- Idea: a 2-phase process
  - Phase 1: guessing whether defaults’ conclusions belong to the extension ($\lambda'$)
  - Phase 2: applying defaults and check if $E$ and $\Gamma_\Delta(E)$ coincide ($\lambda$)
Transformation $\Omega$

- Idea: exploit the guessing phase of ASP, the condition for an interpretation to be an answer set
- hence we need to specify only one rule for each default:

$$\text{aux}_\gamma(\overrightarrow{Z}) \leftarrow DL[\lambda; \alpha](\overrightarrow{X}),$$
$$\text{not } DL[\lambda; \neg \beta_1](\overrightarrow{Y}_1), \ldots, \text{not } DL[\lambda; \neg \beta_m](\overrightarrow{Y}_m).$$

where:

$$\lambda = \bigcup_{\delta_i \in D}(\gamma_i \cup \text{in\_name}(\gamma_i), \gamma_i \cup \text{in\_not\_name}(\gamma_i))$$
The algorithm

1. Select a set of justifications $J \subseteq j(D)$
2. Find the set of defaults $S$ whose justifications belong to $J$
3. Compute the set of consequences $E$ of $W$ that can be derived by means of defaults in $S$ (a default fires if its prerequisite has been derived earlier).
4. If all justifications in $J$ are consistent with $E$ and every default not in $S$ has at least one justification not consistent with $E$, the output $E$ as an extension.
5. Repeat until all subsets of $j(D)$ are considered or pruned.
Transformation $\Upsilon$

- Rules that select justifications:
  
  $auxc_{-\beta_i}(\overrightarrow{Y_i}) \leftarrow \text{not } out_{auxc_{-\beta_i}}(\overrightarrow{Y_i})$.
  
  $out_{auxc_{-\beta_i}}(\overrightarrow{Y_i}) \leftarrow \text{not } auxc_{-\beta_i}(\overrightarrow{Y_i})$.

- A rule which computes the set of consequences $E$:
  
  $aux_{-\gamma}(\overrightarrow{Z}) \leftarrow DL[\lambda; \alpha](\overrightarrow{X}), auxc_{-\beta_1}(\overrightarrow{Y_1}), \ldots, auxc_{-\beta_m}(\overrightarrow{Y_m})$.

where:

$\lambda = \bigcup_{\delta_i \in D}(\gamma_i \cup in\_name(\gamma_i), \gamma_i \cup in\_not\_name(\gamma_i))$
Transformation $\Upsilon$ - cont.

- Rules that check the compliance of our guess with $E$

\[
\text{fail} \leftarrow DL[\lambda; \neg \beta_i](\overrightarrow{Y}_i), auxc_\beta(\overrightarrow{Y}_i), \text{ not fail}.
\]

\[
\text{fail} \leftarrow \text{not } DL[\lambda; \neg \beta_i](\overrightarrow{Y}_i), out_{auxc}_\beta(\overrightarrow{Y}_i), \text{ not fail}.
\]
3 transformations were tested under different examples: Tweety bird, Nixon Diamond, Small Wine, etc., and two running modes, namely using caching and not.

Criteria to compare: total running time, RacerPro’s time and dlvhex time.

We show the result of the Tweety bird example.
Motivating Example

Transformation π

<table>
<thead>
<tr>
<th>Number of individuals</th>
<th>total time (query caching on)</th>
<th>racer time (query caching on)</th>
<th>total time (query caching off)</th>
<th>racer time (query caching off)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

DAO Tran Minh Vienna University of Technology

Default Reasoning on Top of Ontologies with dl-Programs
Motivating Example

Introduction

Transformations

Experimental Results

Conclusions and Future work

Transformation $\Omega$

![Graph showing evaluation time vs. number of individuals for different query caching settings.](image)

- **total time (query caching on)**
- **racer time (query caching on)**
- **total time (query caching off)**
- **racer time (query caching off)**

DAO Tran Minh

Vienna University of Technology

Default Reasoning on Top of Ontologies with dl-Programs
Transformation $\Upsilon$

**Motivating Example**

**Introduction**

**Transformations**

**Experimental Results**

**Conclusions and Future work**

**Evaluation Time / secs**

- total time (query caching on)
- racer time (query caching on)
- total time (query caching off)
- racer time (query caching off)

**Number of individuals**

DAO Tran Minh

Vienna University of Technology

Default Reasoning on Top of Ontologies with dl-Programs
Compare 3 transformations in caching mode

The graph shows the evaluation time in seconds for different numbers of individuals. The x-axis represents the number of individuals, ranging from 1 to 11. The y-axis represents the evaluation time in seconds, ranging from 0.01 to 1,000. The graph includes three lines, each representing a different transformation (denoted as Π, Ω, and Υ). The lines indicate the trend of increasing evaluation time as the number of individuals increases.
Conclusions:

- Three transformations work correctly
- \( \Omega \) and \( \Upsilon \) are much faster than \( \Pi \)
- Caching technique concerning calls to ontologies plays an important role in improving the system's performance

Future work:

- Investigate more pruning rules
- Upgrade \texttt{dlvhex} for pruning rules to take effect
- Investigate transformations for special default theories such as normal default, semi-normal default
- Implement caching for cq-programs in the \texttt{dl-plugin}
- Interface to different DL-reasoners, eg., Pellet, KAON2
- Explore the possibility of classifying the input to reduce the search space