Default Reasoning on Top of Ontologies with dl-Programs

DAO Tran Minh
Supervisor: Prof. Thomas Eiter

Vienna University of Technology

June 12, 2008

Outline

- Simple bird ontology
  - $Flier \sqsubseteq \neg NonFlier$
  - $Penguin \sqsubseteq Bird$
  - $Penguin \sqsubseteq NonFlier$
  - $Penguin(tweety)$
  - $Bird(joe)$
- 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

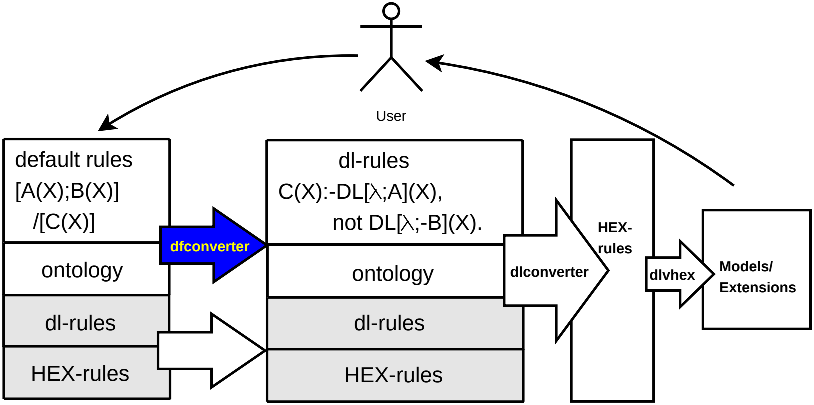
- One of the most famous nonmonotonic reasoning formalizations.
- Default rules: $\frac{\alpha(\overrightarrow{X}):\beta_1(\overrightarrow{X}),\ldots,\beta_m(\overrightarrow{X})}{\gamma(\overrightarrow{X})}$
- Default theory: $T = \langle W, D \rangle$.
- The totality of knowledge induced by a default theory: extension.
- Our purpose: allow each α, β, γ to be either a concept or a role name in a DL-KB. For instance:

$$\frac{Bird(X) : Flier(X)}{Flier(X)}$$

- 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:

$$\mathtt{DL}[\underbrace{\mathtt{WhiteWine} \uplus \mathtt{iswhitewhine}}_{\textit{input list for updating}}; \underbrace{\neg\mathtt{WhiteWine}}_{\textit{query}}](\mathtt{X}).$$

  - strict semantics integration
- Pratical 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


User
default rules
[A(X);B(X)]
/[C(X)]
ontology
dl-rules
HEX-rules
dfconverter
dl-rules
C(X):-DL[λ;A](X),
not DL[λ;-B](X).
ontology
dl-rules
HEX-rules
dlconverter
HEX-
rules
dlvhex
Models/
Extensions

Some conventions

- Default theory $\Delta = \langle L, D \rangle$; L is a DL knowledge base, $D \equiv \{\delta_1, \ldots, \delta_n\}$
- $\delta \equiv \frac{\alpha(\overrightarrow{X}):\beta_1(\overrightarrow{Y}_1),\cdots,\beta_m(\overrightarrow{Y}_m)}{\gamma(\overrightarrow{Z})}$
- $name(\gamma)$: predicate name of the literal γ
- aux_γ:
  - $in\_name(\gamma)$ if γ is positive
  - $in\_not\_name(\gamma)$ if γ is negative
- $auxc\_\beta_i$:
  - $cons\_name(\beta_i)$ if β_i is positive
  - $cons\_not\_name(\beta_i)$ if β_i is negative

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
  $fail \leftarrow DL[\lambda'; \gamma](\overrightarrow{Z}), out\_aux\_\gamma_i(\overrightarrow{Z}), \text{ not } fail.$
  where $\lambda' \equiv \bigcup_{\delta_i \in D}(\gamma_i \uplus in\_name(\gamma_i); \gamma_i \uplus in\_not\_name(\gamma_i))$
- 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 $\lambda \equiv \bigcup_{\delta_i \in D}(\gamma_i \uplus p\_in\_name(\gamma_i); \gamma_i \uplus p\_in\_not\_name(\gamma_i))$

Transformation Π - cont.

- Rules which check whether E and $\Gamma_\Delta(E)$ coincide:
  $fail \leftarrow \text{ not } DL[\lambda;\gamma](\overrightarrow{Z}), aux\_\gamma(\overrightarrow{Z}), \text{ not } fail.$
  $fail \leftarrow DL[\lambda;\gamma](\overrightarrow{Z}), out\_aux\_\gamma(\overrightarrow{Z}), \text{ not } fail.$
- Idea: a 2-phase process
  - Phase 1: guessing whether defaults' conclusions belong to the extension (λ')
  - Phase 2: applying defaults and check if E and $\Gamma_\Delta(E)$ coincide (λ)

Transformation Ω

- 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:
- $aux\_\gamma(\overrightarrow{Z}) \leftarrow DL[\lambda; \alpha](\overrightarrow{X}),$
  $\quad \text{not } DL[\lambda; \neg\beta_1](\overrightarrow{Y}_1), \ldots, \text{ not } DL[\lambda; \neg\beta_m](\overrightarrow{Y}_m).$
  where:
  $\lambda = \bigcup_{\delta_i \in D}(\gamma_i \uplus in\_name(\gamma_i), \gamma_i \cup\!\!\!\!- in\_not\_name(\gamma_i))$

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 Υ

- 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:
  $\lambda = \bigcup_{\delta_i \in D}(\gamma_i \uplus in\_name(\gamma_i), \gamma_i \cup\!\!\!\!- in\_not\_name(\gamma_i))$

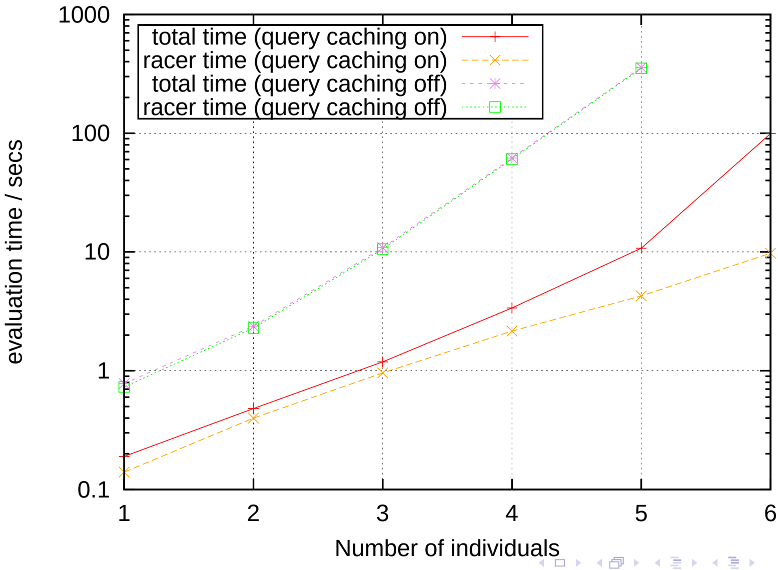
Transformation Υ - cont.

- Rules that check the compliance of our guess with E
  $fail \leftarrow DL[\lambda; \neg\beta_i](\overrightarrow{Y}_i), auxc\_\beta_i(\overrightarrow{Y}_i),\ \text{not}\ fail.$

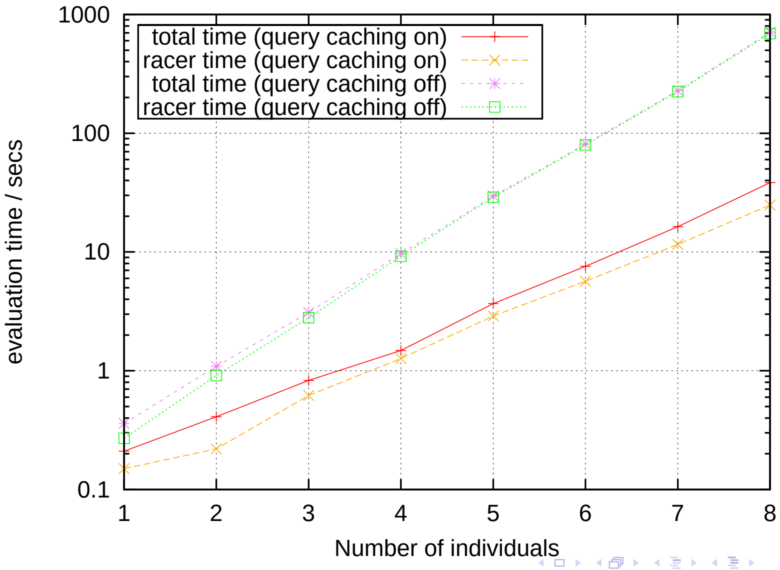
  $fail \leftarrow\ \text{not}\ DL[\lambda; \neg\beta_i](\overrightarrow{Y}_i), out\_auxc\_\beta_i(\overrightarrow{Y}_i),\ \text{not}\ fail.$

- 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

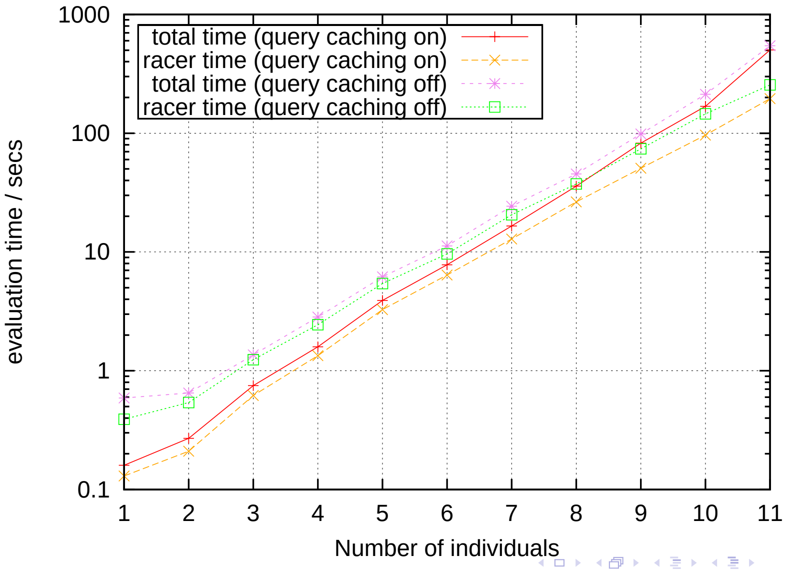
Transformation Π



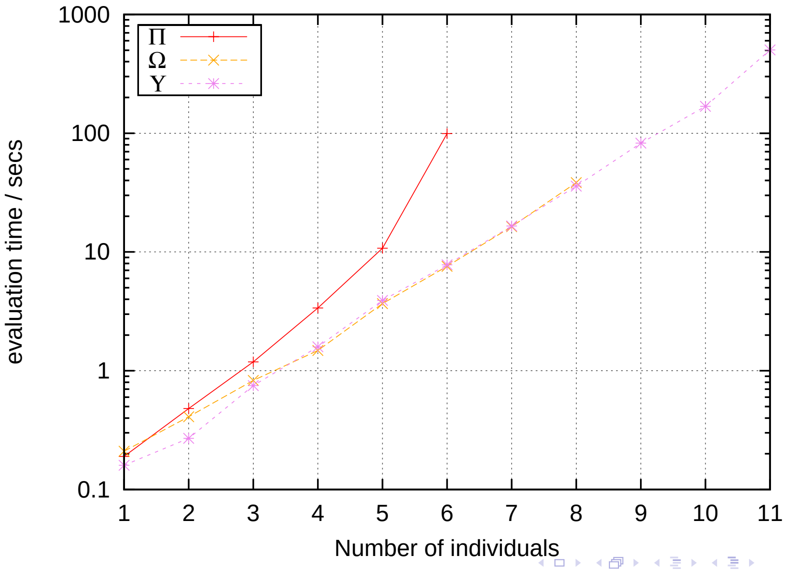
Transformation Ω



Transformation Υ



Compare 3 transformations in caching mode



- Conclusions:
  - Three transformations work correctly
  - Ω and Υ are much faster than Π
  - Caching technique concerning calls to ontologies plays an important role in improving the system's performance
- Future work:
  - Investigate more pruning rules
  - Upgrade `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 `dl-plugin`
  - Interface to different DL-reasoners, eg., Pellet, KAON2
  - Explore the possibility of classifying the input to reduce the search space