Mapping Data to Ontologies With Exceptions Using Answer Set Programming

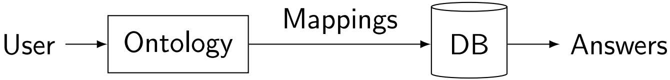
Daniel P. Lupp and Evgenij Thorstensen

11th July 2016

Department of Informatics

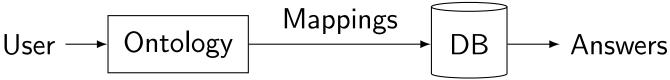
University of Oslo

Ontology-Based Data Access



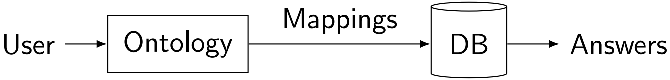
- databases are connected to an ontology using mappings of the form $\varphi \rightsquigarrow \psi$, where φ is a database query and ψ is an ontology query

Ontology-Based Data Access

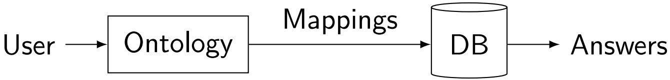


- databases are connected to an ontology using mappings of the form $\varphi \rightsquigarrow \psi$, where φ is a database query and ψ is an ontology query
- In order to query a database, users can phrase queries in the ontology language
- These queries are then translated to database queries using the mappings

Ontology-Based Data Access

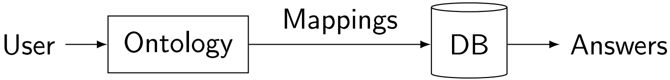


- databases are connected to an ontology using mappings of the form $\varphi \rightsquigarrow \psi$, where φ is a database query and ψ is an ontology query
- In order to query a database, users can phrase queries in the ontology language
- These queries are then translated to database queries using the mappings
- Enables the use of a domain model that closely resembles end-users' understanding of a domain as opposed to complex and convoluted database schemas



- Formally, for $\varphi \rightsquigarrow \psi$:
  Given a database instance $\mathcal{D}$ and a set of mappings $\mathcal{M}$, $\mathcal{I}$ is a model of $(\mathcal{D}, \mathcal{M})$ if $\mathcal{I} \models \psi(\mathbf{t})$ for every query answer $\mathbf{t}$ of φ over $\mathcal{D}$.
- Mapping rewriting of an ontology query ψ w.r.t. $\mathcal{M}$: $\bigvee \varphi_i$ for every i with $\varphi_i \rightsquigarrow \psi \in \mathcal{M}$.

Ontology-Based Data Access



- Formally, for $\varphi \rightsquigarrow \psi$:
  Given a database instance $\mathcal{D}$ and a set of mappings $\mathcal{M}$, $\mathcal{I}$ is a model of $(\mathcal{D}, \mathcal{M})$ if $\mathcal{I} \models \psi(\mathbf{t})$ for every query answer $\mathbf{t}$ of φ over $\mathcal{D}$.
- Mapping rewriting of an ontology query ψ w.r.t. $\mathcal{M}$: $\bigvee \varphi_i$ for every i with $\varphi_i \rightsquigarrow \psi \in \mathcal{M}$.

Example:

$$\texttt{JOBS\_DB}(x, \text{``}\mathit{Accountant}\text{''}) \rightsquigarrow \mathit{Empl}(x)$$
$$\texttt{JOBS\_DB}(x, \text{``}\mathit{IT}\text{''}) \rightsquigarrow \mathit{Empl}(x)$$

then $\mathit{Empl}(x)$ would be rewritten to $\texttt{JOBS\_DB}(x, \text{``}\mathit{IT}\text{''}) \vee \texttt{JOBS\_DB}(x, \text{``}\mathit{Accountant}\text{''})$.

Limitations - OBDA

- databases typically use *closed-world (CW) reasoning*: if data cannot be explicitly found in the database, it is assumed to be false.

Person	Name
	Alice
	Bob
	Carla

$\rightarrow$ John is not a person, i.e., $\neg Person(John)$ is true.

Limitations - OBDA

- databases typically use *closed-world (CW) reasoning*: if data cannot be explicitly found in the database, it is assumed to be false.

Person	Name
	Alice
	Bob
	Carla

$\rightarrow$ John is not a person, i.e., $\neg Person(John)$ is true.

- ontologies employ *open-world (OW) reasoning*, where, in the above example, $\neg Person(John)$ could be either true or false

Limitations - OBDA

- databases typically use *closed-world (CW) reasoning*: if data cannot be explicitly found in the database, it is assumed to be false.

Person	Name
	Alice
	Bob
	Carla

$\rightarrow$ John is not a person, i.e., $\neg Person(John)$ is true.

- ontologies employ *open-world (OW) reasoning*, where, in the above example, $\neg Person(John)$ could be either true or false
- mapping assertions $\varphi \rightsquigarrow \psi$ are interpreted as first-order implications, and thus inherently open-world!

Limitations - OBDA

- no support for ontology constraints over the data, e.g., “every instance of *Person* must be in the table `JOBS_DB`”.

Limitations - OBDA

- no support for ontology constraints over the data, e.g., "every instance of *Person* must be in the table `JOBS_DB`".
- poor handling of exceptions in mappings: must be named explicitly in each mapping, since these are first-order. $\rightarrow$ difficult to maintain and prone to error!

Limitations - OBDA

- no support for ontology constraints over the data, e.g., "every instance of *Person* must be in the table `JOBS_DB`".
- poor handling of exceptions in mappings: must be named explicitly in each mapping, since these are first-order. $\rightarrow$ difficult to maintain and prone to error!

  $\rightarrow$ nonmonotonicity

Nonmonotonic Extensions

- adding nonmonotonicity to OBDA and description logic ontologies is an ongoing research topic, e.g., DL-programs [EIL+08], hybrid-MKNF knowledge bases [DNR02, MR10], closed predicates [LSW13].
- However, the focus is on adding these capabilities to the ontologies

Nonmonotonic Extensions

- adding nonmonotonicity to OBDA and description logic ontologies is an ongoing research topic, e.g., DL-programs [EIL$^+$08], hybrid-MKNF knowledge bases [DNR02, MR10], closed predicates [LSW13].
- However, the focus is on adding these capabilities to the ontologies
- propose extending mappings instead, as they are the tool used to connect closed-world and open-world

$\rightarrow$ *mapping programs*, an extension of $\exists$-ASP [GGLS15]

∃-ASP

An extension of classical ASP that supports

- existential quantification in the heads and negative bodies of rules,
- conjunctive queries in the heads and negative bodies of rules

An ∃-rule is of the form

$$H_1, \ldots, H_n \leftarrow B_1, \ldots, B_m, \\ \textbf{not}\,(C_1^1, \ldots, C_{u_1}^1), \ldots, \textbf{not}\,(C_1^s, \ldots, C_{u_s}^s).$$

where the H_i, B_j, C_k^l are atoms.

- all variables not occuring in the positive body are interpreted existentially.

∃-ASP

- due to the presence of existentials, only variables that are not existentials in negative bodies are grounded (*partial grounding*)
- the existential variables in the rule heads are *Skolemized*
- then the reduct and ∃-answer sets are defined anaologously to their classical ASP counterparts

∃-ASP

- due to the presence of existentials, only variables that are not existentials in negative bodies are grounded (*partial grounding*)
- the existential variables in the rule heads are *Skolemized*
- then the reduct and ∃-answer sets are defined anaologously to their classical ASP counterparts

Theorem ([GGLS15])

For a given ∃-ASP program there exists an equivalent (w.r.t. answer sets) classical ASP program.

$\rightarrow$ reasoning in ∃-ASP can be reduced to reasoning in ASP

Mapping Programs

Extends ∃-ASP to allow for ontology queries in rule bodies

- A mapping rule is of the form

$$m : H^{\mathcal{T}}(\mathbf{x}, \mathbf{z}) \leftarrow \mathbf{not}\, J_1^-(\mathbf{y}_1), \ldots, \mathbf{not}\, J_k^-(\mathbf{y}_k),$$
$$J_1^+(\mathbf{y}'_1), \ldots, J_l^+(\mathbf{y}'_l), Q^{\mathcal{S}}(\mathbf{x}).$$

  where $Q^{\mathcal{S}}$ is a first-order formula over the database and $H^{\mathcal{T}}$, J_i^-, J_j^+ are first-order formulas over the ontology. The variables in $\mathbf{z}$ are existential variables, and $\mathbf{y}_i, \mathbf{y}'_j \subseteq \mathbf{x}$.

Mapping Programs

Extends ∃-ASP to allow for ontology queries in rule bodies

- A mapping rule is of the form

$$m : H^{\mathcal{T}}(\mathbf{x}, \mathbf{z}) \leftarrow \mathbf{not}\ J_1^-(\mathbf{y}_1), \ldots, \mathbf{not}\ J_k^-(\mathbf{y}_k), \\ J_1^+(\mathbf{y}_1'), \ldots, J_l^+(\mathbf{y}_l'), Q^{\mathcal{S}}(\mathbf{x}).$$

  where $Q^{\mathcal{S}}$ is a first-order formula over the database and $H^{\mathcal{T}}$, J_i^-, J_j^+ are first-order formulas over the ontology. The variables in $\mathbf{z}$ are existential variables, and $\mathbf{y}_i, \mathbf{y}_j' \subseteq \mathbf{x}$.

- Intuitively, this can be read as follows:

  $Q^{\mathcal{S}}$ is mapped to $H^{\mathcal{T}}$ if all J_j^+ are certain answers and all J_i^- are not certain answers w.r.t. the mapping and ontology.

  J^+ and J^- are called the *positive* and *negative justifications*, respectively.

Example:
Let $\mathcal{D}$ consist of just one table, `JOBS_DB(<NAME>,<JOB>)`, and $\Sigma_{\mathcal{T}} = \{Empl, hasSup, depHeadOf\}$ be the signature of $\mathcal{T}$. The mapping rule

$$m_1 : \exists Z.hasSup(X, Z) \leftarrow \textbf{not}\, \exists Y.depHeadOf(X, Y), Empl(X), \texttt{Jobs\_DB}(X, P).$$

describes the default rule "employees, of whom we do not know that they are the head of a department, have a supervisor."

Mapping Programs — Skolemization

For a mapping rule

$$m : H^{\mathcal{T}}(\mathbf{x}, \mathbf{z}) \leftarrow \textbf{not}\, J_1^-(\mathbf{y}_1), \ldots, \textbf{not}\, J_k^-(\mathbf{y}_k), J_1^+(\mathbf{y}_1'), \ldots, J_l^+(\mathbf{y}_l'), Q^{\mathcal{S}}(\mathbf{x}).$$

define the *Skolem mapping rule* $sk(m)$ by replacing each existential variable $\mathbf{z}$ in $H^{\mathcal{T}}(\mathbf{x}, \mathbf{z})$ with a Skolem function symbol $sk_{\mathbf{z}}(s)$, where s is an ordered sequence of the universal variables $\mathbf{x}$.

Mapping Programs — Skolemization

For a mapping rule

$$m : H^{\mathcal{T}}(\mathbf{x}, \mathbf{z}) \leftarrow \textbf{not}\ J_1^-(\mathbf{y}_1), \ldots, \textbf{not}\ J_k^-(\mathbf{y}_k), J_1^+(\mathbf{y}'_1), \ldots, J_l^+(\mathbf{y}'_l), Q^{\mathcal{S}}(\mathbf{x}).$$

define the *Skolem mapping rule* $sk(m)$ by replacing each existential variable $\mathbf{z}$ in $H^{\mathcal{T}}(\mathbf{x}, \mathbf{z})$ with a Skolem function symbol $sk_{\mathbf{z}}(s)$, where s is an ordered sequence of the universal variables $\mathbf{x}$.

Definition (Skolem program [GGLS15])

For a mapping program $\mathcal{M}$, the set $sk(\mathcal{M}) = \{sk(m) \mid m \in \mathcal{M}\}$ is called the *Skolem program* of $\mathcal{M}$.

Mapping Programs — Skolemization

For a mapping rule

$$m : H^{\mathcal{T}}(\mathbf{x}, \mathbf{z}) \leftarrow \textbf{not } J_1^-(\mathbf{y}_1), \ldots, \textbf{not } J_k^-(\mathbf{y}_k), J_1^+(\mathbf{y}'_1), \ldots, J_l^+(\mathbf{y}'_l), Q^{\mathcal{S}}(\mathbf{x}).$$

define the *Skolem mapping rule* $sk(m)$ by replacing each existential variable $\mathbf{z}$ in $H^{\mathcal{T}}(\mathbf{x}, \mathbf{z})$ with a Skolem function symbol $sk_{\mathbf{z}}(s)$, where s is an ordered sequence of the universal variables $\mathbf{x}$.

Definition (Skolem program [GGLS15])

For a mapping program $\mathcal{M}$, the set $sk(\mathcal{M}) = \{sk(m) \mid m \in \mathcal{M}\}$ is called the *Skolem program* of $\mathcal{M}$.

Example:

$$\begin{aligned} sk(m_1) : hasSup(X, sk_z(X)) \leftarrow & \textbf{not } \exists Y.depHeadOf(X, Y), \\ & Empl(X), \texttt{Jobs\_DB}(X, P). \end{aligned}$$

Mapping Programs — Partial grounding

Definition (Partial ground programs, analogous to [GGLS15])

The *partial grounding* $PG(m)$ of a mapping rule m is the set of all partial ground instances of m over constants in $\Sigma_{\mathcal{D}}$ for those variables that are not existential variables in the J_i^-. The *partial ground program* of a mapping program $\mathcal{M}$ is the set $PG(\mathcal{M}) = \bigcup_{m \in \mathcal{M}} PG(m)$.

Example:

$$
\begin{aligned}
sk(m_1) : hasSup(X, sk_z(X)) \leftarrow & \textbf{not}\ \exists Y.depHeadOf(X, Y), \\
& Empl(X), \texttt{Jobs\_DB}(X, P).
\end{aligned}
$$

Mapping Programs — Partial grounding

Definition (Partial ground programs, analogous to [GGLS15])

The *partial grounding* $PG(m)$ of a mapping rule m is the set of all partial ground instances of m over constants in $\Sigma_{\mathcal{D}}$ for those variables that are not existential variables in the J_i^-. The *partial ground program* of a mapping program $\mathcal{M}$ is the set $PG(\mathcal{M}) = \bigcup_{m \in \mathcal{M}} PG(m)$.

Example:

$$\begin{aligned} sk(m_1) : hasSup(X, sk_z(X)) \leftarrow & \textbf{not}\ \exists Y.depHeadOf(X, Y), \\ & Empl(X), \texttt{Jobs\_DB}(X, P). \end{aligned}$$

Mapping Programs — Partial grounding

Example:

$$
\begin{aligned}
sk(m_1) : hasSup(X, sk_z(X)) \leftarrow & \textbf{not}\ \exists Y.depHeadOf(X, Y),\\
& Empl(X), \texttt{Jobs\_DB}(X, P).
\end{aligned}
$$

Assume the constants occuring in the database are $\{a, b\}$, then $PG(sk(m_1))$ consists of the four mapping rules

$$
\begin{aligned}
hasSup(a, sk_z(a)) &\leftarrow \textbf{not}\ \exists Y.depHeadOf(a, Y), Empl(a), \texttt{Jobs\_DB}(a, a).\\
hasSup(a, sk_z(a)) &\leftarrow \textbf{not}\ \exists Y.depHeadOf(a, Y), Empl(a), \texttt{Jobs\_DB}(a, b).\\
hasSup(b, sk_z(b)) &\leftarrow \textbf{not}\ \exists Y.depHeadOf(b, Y), Empl(b), \texttt{Jobs\_DB}(b, a).\\
hasSup(b, sk_z(b)) &\leftarrow \textbf{not}\ \exists Y.depHeadOf(b, Y), Empl(b), \texttt{Jobs\_DB}(b, b).
\end{aligned}
$$

Mapping Programs — $\mathcal{T}$-Reduct

Since mapping rules include ontology predicates, we must take the ontology $\mathcal{T}$ into account when constructing the reduct:

Definition ($\mathcal{T}$-reduct)

Given an ontology $\mathcal{T}$, the $\mathcal{T}$-reduct $PG(\mathcal{M})^{\mathcal{A}}$ of a partial ground mapping program $PG(\mathcal{M})$ w.r.t. an interpretation $\mathcal{A}$ is the program obtained from $PG(\mathcal{M})$ after applying the following:

1. Remove all mapping rules m where there exists some $i \leq k$ such that $\mathcal{T} \cup \mathcal{A} \models J_i^-$.
2. Remove all negative justifications from the remaining rules.

Mapping Programs — $\mathcal{T}$-Reduct

Since mapping rules include ontology predicates, we must take the ontology $\mathcal{T}$ into account when constructing the reduct:

Definition ($\mathcal{T}$-reduct)

Given an ontology $\mathcal{T}$, the $\mathcal{T}$-reduct $PG(\mathcal{M})^{\mathcal{A}}$ of a partial ground mapping program $PG(\mathcal{M})$ w.r.t. an interpretation $\mathcal{A}$ is the program obtained from $PG(\mathcal{M})$ after applying the following:

1. Remove all mapping rules m where there exists some $i \leq k$ such that $\mathcal{T} \cup \mathcal{A} \models J_i^-$.
2. Remove all negative justifications from the remaining rules.

$\rightarrow$ the $\mathcal{T}$-reduct is a positive mapping program

Mapping Programs — Semantics

- A *mapping interpretation* $\mathcal{A}$ is a consistent subset of $HB_{sk(\mathcal{M})}$ (Herbrand base over $sk(\mathcal{M})$)
- $\mathcal{A}$ *satisfies the body* of a positive Skolemized mapping rule

$$sk(m) : H^{\mathcal{T}}(\mathbf{x}, sk_{\mathbf{z}}(\mathbf{x})) \leftarrow J_1^+(\mathbf{y}_1'), \ldots, J_l^+(\mathbf{y}_l'), Q^{\mathcal{S}}(\mathbf{x}).$$

  if the following holds: for every query answer $\mathbf{t}$ of $Q^{\mathcal{S}}$ over $\mathcal{D}$, every interpretation I with $I \models \mathcal{T} \cup \mathcal{A}$ satisfies $J_j^+[\mathbf{t}]$ for all $j \leq l$.
- $\mathcal{A} \models sk(m)$ if $\mathcal{A}$ satisfies the head or does not satisfy the body of $sk(m)$.

Mapping Programs — Semantics

- A *mapping interpretation* $\mathcal{A}$ is a consistent subset of $HB_{sk(\mathcal{M})}$ (Herbrand base over $sk(\mathcal{M})$)
- $\mathcal{A}$ *satisfies the body* of a positive Skolemized mapping rule

$$sk(m) : H^{\mathcal{T}}(\mathbf{x}, sk_{\mathbf{z}}(\mathbf{x})) \leftarrow J_1^{+}(\mathbf{y}_1'), \ldots, J_l^{+}(\mathbf{y}_l'), Q^{\mathcal{S}}(\mathbf{x}).$$

  if the following holds: for every query answer $\mathbf{t}$ of $Q^{\mathcal{S}}$ over $\mathcal{D}$, every interpretation I with $I \models \mathcal{T} \cup \mathcal{A}$ satisfies $J_j^{+}[\mathbf{t}]$ for all $j \leq l$.
- $\mathcal{A} \models sk(m)$ if $\mathcal{A}$ satisfies the head or does not satisfy the body of $sk(m)$.

Definition ($\mathcal{T}$-Answer Set)

A mapping interpretation $\mathcal{A} \subseteq HB_{sk(\mathcal{M})}$ is a $\mathcal{T}$*-answer set* of $\mathcal{M}$ if it is a $\subseteq$-minimal model of the reduct $PG(sk(\mathcal{M}))^{\mathcal{A}}$.

Mapping Programs — $\mathcal{T}$-Reduct

Example:

Let $\mathcal{T} = \{Boss \sqsubseteq \exists depHeadOf\}$ and $\mathcal{M}$ consist of

$$m_1 : \exists Z.hasSup(X, Z) \leftarrow \textbf{not}\, \exists Y.depHeadOf(X, Y), Empl(X), \texttt{Jobs\_DB}(X, P).$$

$$m_2 : Boss(X) \leftarrow \texttt{Jobs\_DB}(X, b).$$

$$m_3 : Empl(X) \leftarrow \texttt{Jobs\_DB}(X, P).$$

Mapping Programs — $\mathcal{T}$-Reduct

Example:

Let $\mathcal{T} = \{Boss \sqsubseteq \exists depHeadOf\}$ and $\mathcal{M}$ consist of

$$m_1 : \exists Z.hasSup(X, Z) \leftarrow \textbf{not}\, \exists Y.depHeadOf(X, Y), Empl(X), \texttt{Jobs\_DB}(X, P).$$

$$m_2 : Boss(X) \leftarrow \texttt{Jobs\_DB}(X, b).$$

$$m_3 : Empl(X) \leftarrow \texttt{Jobs\_DB}(X, P).$$

Then for $\mathcal{A} = \{\texttt{Jobs\_DB}(a, b), Empl(a), Boss(a)\}$, the rules

$$hasSup(a, sk_z(a)) \leftarrow \textbf{not}\, \exists Y.depHeadOf(a, Y), Empl(a), \texttt{Jobs\_DB}(a, v).$$

for $v \in \{a, b\}$ are removed from $PG(sk(\mathcal{M}))^{\mathcal{A}}$, since $\mathcal{T} \cup \mathcal{A} \models \exists Y.depHeadOf(a, Y)$.

Mapping Programs — $\mathcal{T}$-Reduct

Example:

Let $\mathcal{T} = \{Boss \sqsubseteq \exists depHeadOf\}$ and $\mathcal{M}$ consist of

$$m_1 : \exists Z.hasSup(X, Z) \leftarrow \textbf{not}\, \exists Y.depHeadOf(X, Y), Empl(X), \texttt{Jobs\_DB}(X, P).$$

$$m_2 : Boss(X) \leftarrow \texttt{Jobs\_DB}(X, b).$$

$$m_3 : Empl(X) \leftarrow \texttt{Jobs\_DB}(X, P).$$

Then for $\mathcal{A} = \{\texttt{Jobs\_DB}(a, b), Empl(a), Boss(a)\}$, the rules

$$hasSup(a, sk_z(a)) \leftarrow \textbf{not}\, \exists Y.depHeadOf(a, Y), Empl(a), \texttt{Jobs\_DB}(a, v).$$

for $v \in \{a, b\}$ are removed from $PG(sk(\mathcal{M}))^{\mathcal{A}}$, since $\mathcal{T} \cup \mathcal{A} \models \exists Y.depHeadOf(a, Y)$.

Then the $\mathcal{T}$-reduct consists of all groundings of:

$$hasSup(b, sk_z(b)) \leftarrow Empl(b), \texttt{Jobs\_DB}(b, Y).$$

$$Boss(X) \leftarrow \texttt{Jobs\_DB}(X, b).$$

$$Empl(X) \leftarrow \texttt{Jobs\_DB}(X, P).$$

Mapping Programs — $\mathcal{T}$-reduct

Example:
Let $\mathcal{T} = \{Boss \sqsubseteq \exists depHeadOf\}$ and $\mathcal{M}$ consist of

$$m_1 : \exists Z.hasSup(X, Z) \leftarrow \textbf{not}\ \exists Y.depHeadOf(X, Y), Empl(X), \texttt{Jobs\_DB}(X, P).$$

$$m_2 : Boss(X) \leftarrow \texttt{Jobs\_DB}(X, b).$$

$$m_3 : Empl(X) \leftarrow \texttt{Jobs\_DB}(X, P).$$

Then for $\mathcal{A} = \{\texttt{Jobs\_DB}(a, b), Empl(a), Boss(a)\}$, the rules $\mathcal{T}$-answer set!

$$hasSup(a, sk_z(a)) \leftarrow \textbf{not}\ \exists Y.depHeadOf(a, Y), Empl(a), \texttt{Jobs\_DB}(a, v).$$

for $v \in \{a, b\}$ are removed from $PG(sk(\mathcal{M}))^{\mathcal{A}}$, since $\mathcal{T} \cup \mathcal{A} \models \exists Y.depHeadOf(a, Y)$.
Then the $\mathcal{T}$-reduct consists of all groundings of:

$$hasSup(b, sk_z(b)) \leftarrow Empl(b), \texttt{Jobs\_DB}(b, Y).$$

$$Boss(X) \leftarrow \texttt{Jobs\_DB}(X, b).$$

$$Empl(X) \leftarrow \texttt{Jobs\_DB}(X, P).$$

Mapping Programs in OBDA

Putting mapping programs into an OBDA context:

Definition (Generalized OBDA)

A *generalized OBDA specification* is a tuple $(\mathcal{D}, \mathcal{M}, \mathcal{T})$ consisting of a database $\mathcal{D}$, a mapping program $\mathcal{M}$, and an ontology $\mathcal{T}$.

Definition (Generalized OBDA semantics)

A tuple $(\mathcal{I}, \mathcal{A})$ consisting of a first-order model $\mathcal{I}$ and a mapping interpretation $\mathcal{A}$ is a model of $(\mathcal{D}, \mathcal{M}, \mathcal{T})$ if it satisfies the following:

1. $\mathcal{I} \models \mathcal{T} \cup \mathcal{A}$,
2. $\mathcal{A}$ is a $\mathcal{T}$-answer set of $\mathcal{M}$.

Mapping Programs

$$m : H^{\mathcal{T}}(\mathbf{x}, \mathbf{z}) \leftarrow \textbf{not}\, J_1^-(\mathbf{y}_1), \ldots, \textbf{not}\, J_k^-(\mathbf{y}_k), J_1^+(\mathbf{y}'_1), \ldots, J_l^+(\mathbf{y}'_l), Q^{\mathcal{S}}(\mathbf{x}).$$

Noteworthy:

- Due to $\mathbf{y}, \mathbf{y}' \subseteq \mathbf{x}$, $Q^{\mathcal{S}}(\mathbf{x})$ acts as a guard. Mapping rules are only applicable to tuples of constants from the database, not existential witnesses generated by mapping heads!

Mapping Programs

$$m : H^{\mathcal{T}}(\mathbf{x}, \mathbf{z}) \leftarrow \textbf{not } J_1^-(\mathbf{y}_1), \ldots, \textbf{not } J_k^-(\mathbf{y}_k), J_1^+(\mathbf{y}'_1), \ldots, J_l^+(\mathbf{y}'_l), Q^{\mathcal{S}}(\mathbf{x}).$$

Noteworthy:

- Due to $\mathbf{y}, \mathbf{y}' \subseteq \mathbf{x}$, $Q^{\mathcal{S}}(\mathbf{x})$ acts as a guard. Mapping rules are only applicable to tuples of constants from the database, not existential witnesses generated by mapping heads!
- the partial grounding is always finite (for finite databases)

Mapping Programs

$$m : H^{\mathcal{T}}(\mathbf{x}, \mathbf{z}) \leftarrow \textbf{not}\ J_1^-(\mathbf{y}_1), \ldots, \textbf{not}\ J_k^-(\mathbf{y}_k), J_1^+(\mathbf{y}'_1), \ldots, J_l^+(\mathbf{y}'_l), Q^{\mathcal{S}}(\mathbf{x}).$$

Noteworthy:

- Due to $\mathbf{y}, \mathbf{y}' \subseteq \mathbf{x}$, $Q^{\mathcal{S}}(\mathbf{x})$ acts as a guard. Mapping rules are only applicable to tuples of constants from the database, not existential witnesses generated by mapping heads!
- the partial grounding is always finite (for finite databases)
- Can express ontology constraints on the database: Let φ be the query to retrieve all tuples that are not in `JOBS_DB`. Then

  $$\perp \leftarrow \mathit{Person}(X), \varphi(X).$$

  expresses "All instances of *Person* must be contained in the table `JOBS_DB`."

Complexity

$$m : H^{\mathcal{T}}(\mathbf{x}, \mathbf{z}) \leftarrow \textbf{not}\, J_1^-(\mathbf{y}_1), \ldots, \textbf{not}\, J_k^-(\mathbf{y}_k), J_1^+(\mathbf{y}'_1), \ldots, J_l^+(\mathbf{y}'_l), Q^{\mathcal{S}}(\mathbf{x}).$$

In general, mapping programs are extremely expressive: arbitrary first-order formulas $H^{\mathcal{T}}, J^-, J^+$.

Complexity

$$m : H^{\mathcal{T}}(\mathbf{x}, \mathbf{z}) \leftarrow \textbf{not}\, J_1^-(\mathbf{y}_1), \ldots, \textbf{not}\, J_k^-(\mathbf{y}_k), J_1^+(\mathbf{y}'_1), \ldots, J_l^+(\mathbf{y}'_l), Q^{\mathcal{S}}(\mathbf{x}).$$

In general, mapping programs are extremely expressive: arbitrary first-order formulas $H^{\mathcal{T}}, J^-, J^+$.

Theorem

The problem of checking $\mathcal{M} \models A$ for a given mapping program $\mathcal{M}$ and a ground atom A is undecidable.

Complexity

- Let $(\mathcal{T}, \mathcal{L})$ consist of an ontology $\mathcal{T}$ and a set $\mathcal{L}$ of formulas such that $\mathcal{T}$-entailment of any $\varphi \in \mathcal{L}$ is decided by an oracle $\mathcal{O}_{(\mathcal{T},\mathcal{L})}$.

Complexity

- Let $(\mathcal{T}, \mathcal{L})$ consist of an ontology $\mathcal{T}$ and a set $\mathcal{L}$ of formulas such that $\mathcal{T}$-entailment of any $\varphi \in \mathcal{L}$ is decided by an oracle $\mathcal{O}_{(\mathcal{T},\mathcal{L})}$.
- Restrict $H^{\mathcal{T}}, J^{-}, J^{+}$ to formulas from $\mathcal{L}$.

Complexity

- Let $(\mathcal{T}, \mathcal{L})$ consist of an ontology $\mathcal{T}$ and a set $\mathcal{L}$ of formulas such that $\mathcal{T}$-entailment of any $\varphi \in \mathcal{L}$ is decided by an oracle $\mathcal{O}_{(\mathcal{T},\mathcal{L})}$.
- Restrict $H^{\mathcal{T}}, J^{-}, J^{+}$ to formulas from $\mathcal{L}$.
- Then for a partially ground Skolem program $\mathcal{M}$, a guess-and-check algorithm can be used for $\mathcal{T}$-answer set construction: guess $\mathcal{A}$, construct the $\mathcal{T}$-reduct $\mathcal{M}^{\mathcal{A}}$, check satisfiability and minimality of $\mathcal{A}$.

Complexity

- Let $(\mathcal{T}, \mathcal{L})$ consist of an ontology $\mathcal{T}$ and a set $\mathcal{L}$ of formulas such that $\mathcal{T}$-entailment of any $\varphi \in \mathcal{L}$ is decided by an oracle $\mathcal{O}_{(\mathcal{T},\mathcal{L})}$.
- Restrict $H^{\mathcal{T}}, J^{-}, J^{+}$ to formulas from $\mathcal{L}$.
- Then for a partially ground Skolem program $\mathcal{M}$, a guess-and-check algorithm can be used for $\mathcal{T}$-answer set construction: guess $\mathcal{A}$, construct the $\mathcal{T}$-reduct $\mathcal{M}^{\mathcal{A}}$, check satisfiability and minimality of $\mathcal{A}$.
- generalization of the classical ASP guess-and-check: for $\mathcal{T} = \emptyset$ and $\mathcal{L} = \{\text{ground atoms}\}$, mapping programs are precisely ASP programs.

Complexity

- Let $(\mathcal{T}, \mathcal{L})$ consist of an ontology $\mathcal{T}$ and a set $\mathcal{L}$ of formulas such that $\mathcal{T}$-entailment of any $\varphi \in \mathcal{L}$ is decided by an oracle $\mathcal{O}_{(\mathcal{T},\mathcal{L})}$.
- Restrict $H^{\mathcal{T}}, J^{-}, J^{+}$ to formulas from $\mathcal{L}$.
- Then for a partially ground Skolem program $\mathcal{M}$, a guess-and-check algorithm can be used for $\mathcal{T}$-answer set construction: guess $\mathcal{A}$, construct the $\mathcal{T}$-reduct $\mathcal{M}^{\mathcal{A}}$, check satisfiability and minimality of $\mathcal{A}$.
- generalization of the classical ASP guess-and-check: for $\mathcal{T} = \emptyset$ and $\mathcal{L} = \{\text{ground atoms}\}$, mapping programs are precisely ASP programs. $\rightarrow$ at least NP-hard (data complexity)

Complexity

More generally:

Theorem

Let $(\mathcal{T}, \mathcal{L})$ be a pair consisting of an first-order ontology $\mathcal{T}$ and a set of formulas $\mathcal{L}$ over the language of $\mathcal{T}$ such that $\mathcal{T}$-entailment is $|\mathcal{O}_{(\mathcal{T},\mathcal{L})}|$-hard for an oracle $\mathcal{O}_{(\mathcal{T},\mathcal{L})}$. Then for a partially ground Skolemized mapping program $\mathcal{M}$ where $H^{\mathcal{T}}, J^{-}, J^{+}$ are from $\mathcal{L}$, $\mathcal{T}$-answer set existence is $NP^{\mathcal{O}_{(\mathcal{T},\mathcal{L})}}$-complete.

Complexity

More generally:

Theorem

Let $(\mathcal{T}, \mathcal{L})$ be a pair consisting of an first-order ontology $\mathcal{T}$ and a set of formulas $\mathcal{L}$ over the language of $\mathcal{T}$ such that $\mathcal{T}$-entailment is $|\mathcal{O}_{(\mathcal{T},\mathcal{L})}|$-hard for an oracle $\mathcal{O}_{(\mathcal{T},\mathcal{L})}$. Then for a partially ground Skolemized mapping program $\mathcal{M}$ where $H^{\mathcal{T}}, J^{-}, J^{+}$ are from $\mathcal{L}$, $\mathcal{T}$-answer set existence is $NP^{\mathcal{O}_{(\mathcal{T},\mathcal{L})}}$-complete.

Proof idea:
Universal reduction argument: An $\mathrm{NP}^{\mathcal{O}_{(\mathcal{T},\mathcal{L})}}$ Turing machine can be encoded as a mapping program in the same manner an NP TM can be encoded in ASP, but allowing for oracle calls in rule bodies.

UCQ-Rewritability — Reduction to ASP

- The $\mathcal{T}$*-rewriting* of a query φ is a query $\overline{\varphi}$ such that
$$(\mathcal{D}, \mathcal{M}, \mathcal{T}) \models \varphi \text{ iff } (\mathcal{D}, \mathcal{M}, \emptyset) \models \overline{\varphi}.$$
- A query φ is *UCQ-rewritable* if its $\mathcal{T}$-rewriting is a union of conjunctive queries

UCQ-Rewritability — Reduction to ASP

- The $\mathcal{T}$*-rewriting* of a query φ is a query $\overline{\varphi}$ such that
$$(\mathcal{D}, \mathcal{M}, \mathcal{T}) \models \varphi \text{ iff } (\mathcal{D}, \mathcal{M}, \emptyset) \models \overline{\varphi}.$$
- A query φ is *UCQ-rewritable* if its $\mathcal{T}$-rewriting is a union of conjunctive queries
- Let $\mathcal{M}$ contain only rules where J^+, J^- are UCQ-rewritable w.r.t. $\mathcal{T}$ and $H^{\mathcal{T}}$ are conjunctive queries.

UCQ-Rewritability — Reduction to ASP

- The $\mathcal{T}$*-rewriting* of a query φ is a query $\overline{\varphi}$ such that
$$(\mathcal{D}, \mathcal{M}, \mathcal{T}) \models \varphi \text{ iff } (\mathcal{D}, \mathcal{M}, \emptyset) \models \overline{\varphi}.$$
- A query φ is *UCQ-rewritable* if its $\mathcal{T}$-rewriting is a union of conjunctive queries
- Let $\mathcal{M}$ contain only rules where J^+, J^- are UCQ-rewritable w.r.t. $\mathcal{T}$ and $H^\mathcal{T}$ are conjunctive queries.
- Define $\overline{\mathcal{M}}$ as the program obtained by replacing J^+, J^- with their $\mathcal{T}$-rewritings $\overline{J^+}$ and $\overline{J^-}$.
- $\overline{\mathcal{M}}$ is equivalent to a $\exists$-program! (standard logic programming transformations)

UCQ-Rewritability — Reduction to ASP

Theorem

For a generalized OBDA specification $(\mathcal{D}, \mathcal{M}, \mathcal{T})$, where J^+, J^- in $\mathcal{M}$ are UCQ-rewritable with respect to $\mathcal{T}$, there exists an $\exists$-ASP program $\mathcal{M}'$ such that for a query q over $\mathcal{T}$

$$(\mathcal{D}, \mathcal{M}, \mathcal{T}) \vDash q[t] \iff \mathcal{M}' \vDash \overline{q}[t],$$

i.e., query answering over $(\mathcal{D}, \mathcal{M}, \mathcal{T})$ reduces to cautious reasoning over $\mathcal{M}'$.

Proof idea: Straightforward calculation.

UCQ-Rewritability — Reduction to ASP

Theorem

For a generalized OBDA specification $(\mathcal{D}, \mathcal{M}, \mathcal{T})$, where J^+, J^- in $\mathcal{M}$ are UCQ-rewritable with respect to $\mathcal{T}$, there exists an $\exists$-ASP program $\mathcal{M}'$ such that for a query q over $\mathcal{T}$

$$(\mathcal{D}, \mathcal{M}, \mathcal{T}) \models q[t] \iff \mathcal{M}' \models \overline{q}[t],$$

i.e., query answering over $(\mathcal{D}, \mathcal{M}, \mathcal{T})$ reduces to cautious reasoning over $\mathcal{M}'$.

Proof idea: Straightforward calculation.
[GGLS15] $\Rightarrow$ can be further reduced to ASP.

Summary and Future Work

Summary:

- mapping programs as new mapping framework based on an extension of $\exists$-ASP.
- supports default exception handling and ontology constraints
- Algorithm for general, decidable case
- Reasoning reduces to ASP if the mapping program is UCQ-rewritable.

Future Work:

- analyze mapping programs from a parameterized complexity perspective
- determine which fragments of mapping programs admit a query rewriting process (currently not possible)
- Analyze when mapping programs can be rewritten to a classical OBDA mapping
- proof-of-concept implementation of an OBDA system using mapping programs

Bibliography

Francesco M. Donini, Daniele Nardi, and Riccardo Rosati.
Description logics of minimal knowledge and negation as failure.
ACM Trans. Comput. Logic, 3(2):177–225, April 2002.

Thomas Eiter, Giovambattista Ianni, Thomas Lukasiewicz, Roman Schindlauer, and Hans Tompits.
Combining answer set programming with description logics for the semantic web.
Artificial Intelligence, 172(12–13):1495 – 1539, 2008.

Fabien Garreau, Laurent Garcia, Claire Lefèvre, and Igor Stéphan.
∃-ASP.
In *Proceedings of the Joint Ontology Workshops 2015 Episode 1: The Argentine Winter of Ontology co-located with the 24th International Joint Conference on Artificial Intelligence (IJCAI 2015), Buenos Aires, Argentina, July 25-27, 2015.*, 2015.

Carsten Lutz, İnanç Seylan, and Frank Wolter.
Ontology-based data access with closed predicates is inherently intractable (sometimes).
In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI '13, pages 1024–1030. AAAI Press, 2013.

Boris Motik and Riccardo Rosati.
Reconciling description logics and rules.
J. ACM, 57(5):30:1–30:62, June 2010.