Actions and programs over Description Logic Ontologies

Diego Calvanese, Giuseppe De Giacomo,
Maurizio Lenzerini, Riccardo Rosati
Dipartimento di Informatica e Sistemistica
Sapienza Università di Roma

INFINT 2009, Bertinoro
Our basic ingredients

**Description Logics (DL)**
- formalisms of choice for modeling **ontologies**: i.e., conceptualizations of the domain of interest
- represent **static aspects** of knowledge: classes, relationships between classes, ISAs, etc.

**Atomic actions over ontologies**
- **queries** (retrieve information)
- **update operations** (add and delete information).

**High-level programs** (à la Congolog)
- deal with **dynamic aspects**: action effects, change, knowledge evolution...
- **high-level descriptions of computations** that abstract from the technological issues of the actual programs that realize them
Our data and service model

Data model = DL ontology
- very rich
- incomplete information/open-world assumption

Service/process description = high-level program
- rich language: sequential composition, if-then-else, while, ...
- atomic actions: read (query) and write (update) the DL ontology

Execution model = state transition system
- state = state of the DL ontology
- transition = execution of an atomic action (update)
- successor state = evolution of the DL ontology (result of an update action)
Example

consider the following (DL) ontology on companies and grants:

∃owns ⊑ Company
∃owns⁻ ⊑ Company
PublicCompany ⊑ Company
PrivateCompany ⊑ Company
∃GrantAsked ⊑ ResearchGroup
∃GrantAsked⁻ ⊑ Company
IllegalOwner ⊑ Company
Example

given a research group \( r \) and a company \( c \), interactively (*) select a public company owned by \( c \) to ask a grant to; if \( c \) does not own public companies, then select the company \( c \) itself:

\[
\text{askNewGrant}(r,c) = \\
\begin{cases} 
  \text{IF (q() \gets \text{owns}(c,y), \text{PublicCompany}(y)) \text{ THEN } \{} \\
  \quad \text{PICK (PublicCompaniesOwnedBy}(c)) \ {\text{ApplyForGrant}(r,x)} \\
  \text{\} } \\
  \text{ELSE ApplyForGrant}(r,c) 
\end{cases}
\]

where:

\[
\text{PublicCompaniesOwnedBy}(c) = q(x) \gets \text{owns}(c,x), \text{PublicCompany}(x)
\]

\[
\text{ApplyForGrant}(r,c) = \text{update GrantAsked}(r,c) \text{ where true}
\]

(*) through a suitable choice function for CHOICE that presents the result of the query to the client, who chooses the tuple s/he is interested in
Combining DL ontologies and actions

Programme 1: work at the level of models

The natural way to do it: *exploit over 30 years of research on Reasoning about Actions*:

- Choose an expressive formalism such as the **Situation Calculus**
- Translate the DL **ontology** (which is essentially a theory expressed in a fragment of FOL) in (e.g.) SitCalc
- Specify **actions** in (e.g.) SitCalc
- Use the **single theory** obtained in this way to represent and reason on actions over the ontology
- Exploit **high-level programming languages** such as Golog/ConGolog developed in AI for formalizing web services
- very **ambitious** yet very **difficult** way of combining DLs and actions (decidability/undecidability results [Artale et al.; Baader et al., KR’08])
Combining DL ontologies and actions

Programme 2: work at the meta-theoretic level

Adopt a radical solution: assume a **functional view of ontologies**

under this view, ontologies are systems that allow for two kinds of operations:

- **ASK**(q, s), which returns the answers to a query q that are logically implied by the ontology s
- **TELL**(a, s), which produces a new ontology s’ as a result of the application of an action a to the ontology s
Combining DL ontologies and actions

Programme 2: work at the meta-theoretic level (cont.)

Disadvantages:
- we don’t have a single theory anymore for representing and reasoning on actions over ontologies
  - the ontology represents what is known
  - actions change what is known (i.e., the ontology), but they are not represented in the (same) ontology
- we lose the possibility of distinguishing between “knowledge” and “truth”

Major advantage:
- it strongly decouples reasoning on the static knowledge from reasoning on the dynamics of the computations over such knowledge ...
- ... as a result, we can lift to DLs many results developed in Reasoning about Actions (and in Verification) in the years: e.g., we can use high-level programming languages such as variants of Golog/ConGolog to formalize web services.
Combining OWL and actions

The current technology for OWL-like languages is mature for:

- **ASK**: based on
  - logical implication of assertions
  - instance checking/retrieval
  - ontology consistency

- **TELL**: based on
  - syntactic add and delete of assertions + consistency check
  - (but: research in semantic updates [Liu et al., KR’06: Updating Description Logic ABoxes])

Both ASK and TELL are NEXPTIME-complete in OWL-DL

⇒ we look at computationally less expensive DLs (DL-Lite)
Combining OWL and actions

concrete example (we denote by \( s \) the ontology in its current state):

- **TELL**: we may allow for atomic actions of the following form:
  - \texttt{add L(x) where q(x)}
  - \texttt{delete L(x) where q(x)}

  where \( L \) is a set of ABox assertions and \( q \) is an instance retrieval query (instance checking), with the following semantics

  \[
  \text{TELL}([\texttt{add L(x) where q(x)}], s) = s \cup \bigcup_{t \in \text{ASK}(q(x), s)} L(t) \\
  \text{if } \text{Mod}(s \cup \bigcup_{t \in \text{ASK}(q(x), s)} L(t)) \neq \emptyset
  \]

  \[
  \text{TELL}([\texttt{delete L(x) where q(x)}], s) = s - \bigcup_{t \in \text{ASK}(q(x), s)} L(t) \\
  \text{if } \text{Mod}(s - \bigcup_{t \in \text{ASK}(q(x), s)} L(t)) \neq \emptyset, \text{ i.e., always}
  \]

- **ASK**: we may allow for any basic DL reasoning task: logical implication, instance checking/retrieval, conjunctive query answering.

  Also, to check executability of (add) actions, we allow for expressions of the form

  \[
  \text{ASK}([\texttt{executable(a)}], s)
  \]

  i.e., consistency checks of the form:

  \[
  \text{Mod}(s \cup \bigcup_{t \in \text{ASK}(q(x), s)} L(t)) \neq \emptyset
  \]
Combining DL-Lite and actions

The current technology for DL-Lite languages is mature for:

- **ASK**: based on *query answering* of UCQ queries (actually more: see EQL-Lite(UCQ)) and *ontology consistency* (for ASK([executable(a)], s))

- **TELL**: based on *semantic instance level update and erasure* + best *approximation* of the results as a DL-Lite ABox.

Both ASK and TELL can be computed in **PTIME** in the size of the ontology.
Digression 1: The DL-Lite family

DL-Lite is a family of Description Logics (tractable OWL-DL fragments)
- main objectives:
  - allow for very efficient treatment of large ABoxes...
  - ...even for very expressive queries (conjunctive queries)

\[ \text{DL-Lite}_\text{core} = \text{basic DL-Lite language} \]
- main DL-Lite dialects:
  - \[ \text{DL-Lite}_F \] (DL-Lite$_\text{core}$ + role functionality)
  - \[ \text{DL-Lite}_R \] (DL-Lite$_\text{core}$ + role hierarchies)
  - \[ \text{DL-Lite}_A \] (DL-Lite$_F$ + DL-Lite$_R$ + attributes + domains)
  - (the current OWL 2 QL proposal is based on DL-Lite$_R$)
DL-LiteF: syntax

concept expressions:
- atomic concept A
- role domain ∃R
- role range ∃R⁻

role expressions:
- atomic role R
- inverse atomic role R⁻

• DL-LiteF TBox = set of
  - concept inclusions
  - concept disjointness assertions
  - functional assertions (stating that a role is functional)

• DL-LiteF ABox = set of ground atoms, i.e., assertions
  - A(a) with A concept name
  - R(a,b) with R role name
### DL-LiteF ontology: example

**TBox:**
- MALE ⊑ PERSON
- FEMALE ⊑ PERSON
- PERSON ⊑ ∃hasFather
- ∃hasFather⁻ ⊑ MALE
- PERSON ⊑ ∃hasMother
- ∃hasMother⁻ ⊑ FEMALE
- MALE ⊑¬ FEMALE
- funct(hasMother)

**ABox:**
- MALE(Bob), MALE(Paul), FEMALE(Ann),
- hasFather(Paul,Ann), hasMother(Mary,Paul)
Expressiveness of DL-Lite w.r.t. OWL-DL

main expressive limitations of DL-Lite w.r.t. OWL-DL:

1. restricted disjunction:
   • no explicit disjunction
   • binary Horn implications (concept and role inclusions)

2. restricted negation:
   • no explicit negation
   • concept (and role) disjointness

3. restricted existential quantification:
   • e.g., no qualified existential concepts

4. limited role cardinality restrictions:
   • only role functionality allowed
   • not a “real” problem
DL-Lite vs. conceptual data models

- DL-Lite captures a very large subset of the constructs of conceptual data modeling languages (UML class diagrams, E-R)
  
  - e.g., DL-Lite\textsubscript{A} captures almost all the E-R model:
    - entities = concepts
    - binary relationships = roles
    - entity attributes = concept attributes
    - relationship attributes = role attributes
    - cardinality constraints (0,1) = concept inclusions and role functionalities
    - ...

  ⇒ DL-Lite = a simple yet powerful ontology language
**Digression 2: Updating DL ontologies**

- **semantic update** over DL ontologies = takes seriously into account the open-world assumption of DLs
- problem of theory update/revision: \( T \odot U \)
- problematic case: when \( T \cup U \) is an *inconsistent* theory
- our approach adopts Winslett’s semantics for update:
  - **minimal change**: prefer those models of \( U \) that *minimally differ* from the models of the original theory \( T \)
- very nice semantics, but computationally very hard – computing the result of the update is not trivial:
  1. we cannot simply eliminate from the ontology the formulas that cause contradiction
  2. even worse, the result of the update may not be expressible in the DL language
(expressible) Updates over DL ontologies

Example (DL-LiteF):

TBox: \{ \exists \text{WillPlay} \sqsubseteq \text{AvailablePlayer}, \ \text{AvailablePlayer} \sqsubseteq \text{Player},
\text{Injured} \sqsubseteq \neg \text{AvailablePlayer} \}\n
ABox: \{ \text{WillPlay}(\text{John}, \text{Allstargame09}) \}\n
The above ontology implies:
\text{AvailablePlayer}(\text{John})
\text{Player}(\text{John})
\neg \text{Injured}(\text{John})

update: \text{Injured}(\text{John})

the result of the update (Winslett’s semantics) can be expressed by the following ABox:

\{ \text{Injured}(\text{John}), \ \text{Player}(\text{John}) \}\n
Notice: John remains a player, and this would not be captured by simply removing \text{WillPlay}(\text{John}, \text{Allstargame09}) from the ABox
Non-expressible updates over DL ontologies

Example of non-expressibility of the update (DL-Lite$_F$):

TBox: $\{ \text{ActivePlayer} \sqsubseteq \exists \text{WillPlay}, \quad \exists \text{WillPlay}^- \sqsubseteq \text{Game},$
\[ \quad \exists \text{WillPlay} \sqsubseteq \neg \text{Injured} \} \]

ABox: $\{ \text{ActivePlayer}(\text{John}) \}$

The above ontology implies:

$\exists x . \text{WillPlay}(\text{John}, x) \land \text{Game}(x)$

$\neg \text{Injured}(\text{John})$

update: $\text{Injured}(\text{John})$

the result of the update cannot be expressed by any DL-Lite$_F$ ABox

( the updated ontology implies the sentence $\exists x . \text{Game}(x)$ )
Instance-level update

- **Instance level update** consist in changes of the ontology ABox only:
  - TBox remains immutable (no schema evolution)
  - ABox can change
- unfortunately the result of update in general cannot be represented as a new primitive ABox in DL-Lite$_F$
- we look for the best approximation of the update: the ontology in DL-Lite$_F$ that has all the models of the update and no other such ontology containing less models exists

**Thm [JLC’09]:** For DL-Lite$_F$ the best approximation always exists and is unique.

**Thm [JLC’09]:** For DL-Lite$_F$ the best approximation can be computed in PTIME in the size of the ontology.
Actions over DL ontologies: atomic actions

we consider atomic actions of the following form:

update \( L(x) \text{ where } q(x) \) (add information)
erase \( L(x) \text{ where } q(x) \) (delete information)

where \( L \) is a set of ABox assertions and \( q \) is an EQL-Lite(UCQ) query, with the following semantics:

\[
\text{TELL}([\text{update } L(x) \text{ where } q(x)], s) = s \circ T \bigcup_{t \in \text{ASK}(q(x), s)} L(t)
\]
if \( \text{Mod}(T \bigcup \bigcup_{t \in \text{ASK}(q(x), s)} L(t)) = \emptyset \)

\[
\text{TELL}([\text{erase } L(x) \text{ where } q(x)], s) = s \bullet T \bigcup_{t \in \text{ASK}(q(x), s)} L(t)
\]
if \( \text{Mod}(T \bigcup \neg \bigcup_{t \in \text{ASK}(q(x), s)} L(t)) = \emptyset \)

where we embedded the approximation in the semantics of update \( \circ T \)
and erasure \( \bullet T \)
We adopt a variant of Golog/ConGolog as formalism for programs: we concentrate on the deterministic fragment (extension to larger languages is easy):

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>action <em>(cf. TELL)</em></td>
</tr>
<tr>
<td>ε</td>
<td>empty sequence of actions</td>
</tr>
<tr>
<td>δ1; δ2</td>
<td>sequential composition</td>
</tr>
<tr>
<td>if φ then δ1 else δ2</td>
<td>if-then-else</td>
</tr>
<tr>
<td>while φ do δ</td>
<td>while</td>
</tr>
<tr>
<td>pick q(x).δ[x]</td>
<td>pick <em>(according to a given choice function)</em></td>
</tr>
</tbody>
</table>
Transition semantics

**Idea:** describe the result of executing a **single step** of a program

- Given a program $\delta$ and an ontology $s$, compute the ontology $s'$ and the program $\delta'$ that remains to be executed after a single step $a$ of $\delta$ in $s$.

  Formally, define the relation $\text{Trans}$, denoted by "$\overset{a}{\longrightarrow}\$":
  $$(\delta, s) \overset{a}{\longrightarrow} (\delta', s')$$

- Assert when a program $\delta$ can be considered **successfully terminated** in an ontology $s$.
  Formally, define a predicate $\text{Final}$, denoted by "$\checkmark$":
  $$(\delta, s) \checkmark$$

$\text{Trans}$ and $\text{Final}$ can be defined inductively in a standard way, using the so-called **transition (structural) rules** [Plotkin81, Nielson&Nielson99]
The structural rules have the following schema:

\[
\begin{array}{c}
\text{CONSEQUENT} \\
\hline
\text{ANTECEDENT}
\end{array}
\text{ if SIDE-CONDITION}
\]

which is to be interpreted logically as:

\[\forall (\text{ANTECEDENT} \wedge \text{SIDE-CONDITION} \rightarrow \text{CONSEQUENT})\]

where:

- \(\forall Q\) stands for the universal closure of all free variables occurring in Q
- \text{ANTECEDENT}, \text{SIDE-CONDITION} and \text{CONSEQUENT} share free variables

Given an ontology and a program, the structural rules define inductively a relation, namely: **the smallest relation satisfying the rules**
Transition rules for programs

\[\begin{align*}
\text{act} : & \quad (a, s) \xrightarrow{a} (\varepsilon, \text{TELL}(a, s)) & \text{if } a \text{ is executable in } s \\
\text{true} & \\
\text{seq} : & \quad (\delta_1 ; \delta_2, s) \xrightarrow{a} (\delta'_1 ; \delta_2, s') \quad (\delta_1 ; \delta_2, s) \xrightarrow{a} (\delta'_2, s') \quad (\delta_1 ; \delta_2, s) \xrightarrow{a} (\delta'_2 ; s') & \text{if } (\delta_1, s) \checkmark \\
& \quad (\delta_1, s) \xrightarrow{a} (\delta'_1, s') \\
\text{if} : & \quad (\text{if } \phi \text{ then } \delta_1 \text{ else } \delta_2, s) \xrightarrow{a} (\delta'_1, s') & \text{if } \text{ASK}(\phi, s) = \text{true} \\
& \quad (\delta_1, s) \xrightarrow{a} (\delta'_1, s') \\
& \quad (\text{if } \phi \text{ then } \delta_1 \text{ else } \delta_2, s) \xrightarrow{a} (\delta'_2, s') & \text{if } \text{ASK}(\phi, s) = \text{false} \\
& \quad (\delta_2, s) \xrightarrow{a} (\delta'_2, s') \\
\text{while} : & \quad (\text{while } \phi \text{ do } \delta, s) \xrightarrow{a} (\delta', \text{while } \phi \text{ do } \delta, s') & \text{if } \text{ASK}(\phi, s) = \text{true} \\
& \quad (\delta, s) \xrightarrow{a} (\delta', s') \\
\text{pick} : & \quad (\text{pick } q(x). \delta[x], s) \xrightarrow{a} (\delta'[\tilde{t}], s') & (\text{for } \tilde{t} = \text{CHOICE}[\text{ASK}(q(x), s)]) \\
& \quad (\delta[\tilde{t}], s) \xrightarrow{a} (\delta'[\tilde{t}], s')
\end{align*}\]
Final rules for programs

\[

c : \frac{(\epsilon, s)^\top}{true} \\
seq : \frac{(\delta_1; \delta_2, s)^\top}{(\delta_1, s)^\top \land (\delta_2; s)^\top}
\]

\[

if : \frac{(\text{if } \phi \text{ then } \delta_1 \text{ else } \delta_2, s)^\top}{(\delta_1, s)^\top} \quad \text{if } \text{ASK}(\phi, s) = true
\]

\[

(\text{if } \phi \text{ then } \delta_1 \text{ else } \delta_2, s)^\top \quad \text{if } \text{ASK}(\phi, s) = false
\]

\[

while : \frac{(\text{while } \phi \text{ do } \delta, s)^\top}{true} \quad \text{if } \text{ASK}(\phi, s) = false
\]

\[

(\text{while } \phi \text{ do } \delta, s)^\top \quad \text{if } \text{ASK}(\phi, s) = true
\]

\[

\text{pick} : \frac{(\text{pick } q(\bar{x}). \delta[\bar{x}], s)^\top}{(\delta[\bar{\bar{t}}], s)^\top} \quad (\text{for } \bar{\bar{t}} = \text{CHOICE}[\text{ASK}(q(\bar{x}), s)])
\]
Example (cont.)

consider the following ontology on companies and grants:

\[\exists \text{owns} \sqsubseteq \text{Company} \]
\[\exists \text{owns}^{-} \sqsubseteq \text{Company} \]
\[\text{PublicCompany} \sqsubseteq \text{Company} \]
\[\text{PrivateCompany} \sqsubseteq \text{Company} \]
\[\exists \text{GrantAsked} \sqsubseteq \text{ResearchGroup} \]
\[\exists \text{GrantAsked}^{-} \sqsubseteq \text{Company} \]
\[\text{IllegalOwner} \sqsubseteq \text{Company} \]
Example (cont.)

populate IllegalOwner with those companies that own themselves, either directly or indirectly:

\[
\text{temp} \text{ is an additional role in the alphabet of the TBox)
\]

```
computeIllegalOwners =
ERASE temp(x1,x2) WHERE q(x1,x2) <- temp(x1,x2);
ERASE IllegalOwner(x) WHERE q(x) <- IllegalOwner(x);
UPDATE temp(x1,x2) WHERE q(x1,x2) <- owns(x1,x2);
WHILE (q() <- K(temp(y1,z), owns(z,y2)), not K(temp(y1,y2)))
DO {
    UPDATE temp(x1,x2) WHERE q(x1,x2) <- K(temp(x1,z), owns(z,x2)), not K(temp(x1,x2));
}
UPDATE IllegalOwner(x) WHERE q(x) <- temp(x,x);
```
Results

**Theorem:** Computing \((\delta, s) \xrightarrow{a} (\delta', s')\) and \((\delta, s)\sqrt{}\) can be done in \(\text{PTIME}\) for \(DL-Lite_F\) ontologies (in \(\text{(N)EXPTIME}\) for OWL-like ontologies)

**Theorem:** Checking whether a sequence of actions \(a_1, \ldots, a_n\) is a (complete/partial) run of a program \(\delta_0\) over an ontology \(s_0\) can be done in \(\text{PTIME}\) for \(DL-Lite_F\) ontologies (in \(\text{(N)EXPTIME}\) for OWL-like ontologies)

**Theorem:** Given a sequence of actions \(a_1, \ldots, a_n\) that is a (complete/partial) run of a program \(\delta_0\) over an ontology \(s_0\), computing the **resulting program** \(\delta_n\) and **ontology** \(s_n\) can be done in \(\text{PTIME}\) for \(DL-Lite_F\) ontologies (in \(\text{(N)EXPTIME}\) for OWL-like ontologies)
Executability and projection

The two classical problems in Reasoning about Actions are:

- **Executability**: check whether a sequence of actions is executable in an ontology
- **Projection**: compute the result of a query in the ontology obtained by executing a sequence of actions in an initial ontology

**Corollary**: *Executability* and *projection* can be solved in PTIME for *DL-Lite*$_F$ ontologies (in (N)EXPTIME for OWL-like ontologies)
Conclusions

Message of this paper: Combining DL(-Lite) ontologies and actions under a functional view of ontologies (ASK and TELL) is already possible.

The approach extends to:

- full Golog, i.e., nondeterminism, procedures
- ConGolog, i.e., concurrency, prioritized interrupts
- IndiGolog, i.e., search, online vs. offline computation
- monitored executions
- interactive/nonterminating programs, often required for web services
- local store, to keep memory of previous results of queries to the ontology (similarly to standard programming languages, such as C or Java)
- nondeterministic atomic actions, i.e., actions that may generate more than one ontology (TELL is a relation, not a function)
Conclusions

What about analysis and synthesis of programs? i.e.:

- verifying executability on every ontology
- verifying termination of a program
- verifying temporal properties on nonterminating programs
- synthesizing (unbounded) plans that achieves a goal (for bounded plans from a give initial ontology we can apply the results above)
- synthesizing a service that fulfills a certain specification

They are still difficult!

- in literature there are good techniques for finite state programs ...
- ... but ontologies are not finite state ...
- ... so, we need forms of abstraction to make the analysis on finite states