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

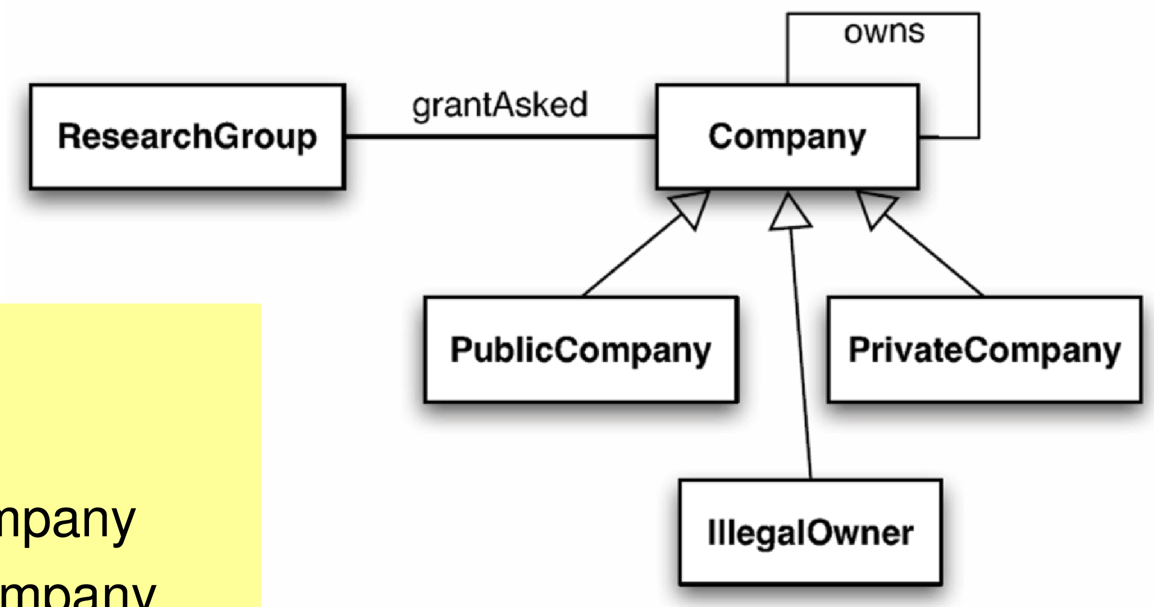
- rich language: sequential composition, if-then-else, while, ...
- atomic actions: read (query) and write (update) the DL ontology

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:



$\exists$owns $\sqsubseteq$ Company
$\exists$owns$^-$ $\sqsubseteq$ Company
PublicCompany $\sqsubseteq$ Company
PrivateCompany $\sqsubseteq$ Company
$\exists$GrantAsked $\sqsubseteq$ ResearchGroup
$\exists$GrantAsked$^-$ $\sqsubseteq$ Company
IllegalOwner $\sqsubseteq$ 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:

```
askNewGrant(r,c) =
IF (q() <- owns(c,y), PublicCompany(y)) THEN {
   PICK (PublicCompaniesOwnedBy(c)) {ApplyForGrant(r,x)}
   }
ELSE ApplyForGrant(r,c)
```

where:

```
PublicCompaniesOwnedBy(c) = q(x) <- owns(c,x), PublicCompany(x)
ApplyForGrant(r,c) = update GrantAsked(r,c) 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** *[Levesque: Foundations of a Functional Approach to Knowledge Representation. AIJ 1984]*

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 (ie, 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

$\Rightarrow$ 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:

  add $L(x)$ **where** $q(x)$

  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

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

  TELL([**delete** $L(x)$ **where** $q(x)$], s) = $s - \bigcup_{t \in \mathrm{ASK}(q(x),s)} L(t)$
  if $\mathrm{Mod}(s - \bigcup_{t \in \mathrm{ASK}(q(x),s)} L(t)) \neq \emptyset$, 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

ASK([executable(a)], s)

i.e., consistency checks of the form: $\mathrm{Mod}(s \cup \bigcup_{t \in \mathrm{ASK}(q(x),s)} L(t)) \neq \emptyset$

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)

DL-Lite$_{core}$ = basic DL-Lite language

- main DL-Lite dialects:
  - **DL-Lite$_F$** (DL-Lite$_{core}$ + role functionality)
  - **DL-Lite$_R$** (DL-Lite$_{core}$ + role hierarchies)
  - **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 $\exists R$
- role range $\exists R^-$

role expressions:
- atomic role R
- inverse atomic role R^-

- DL-Lite$_F$ **TBox** = set of
  - concept inclusions
  - concept disjointness assertions
  - functional assertions (stating that a role is functional)
- DL-Lite$_F$ **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 $\sqsubseteq$ PERSON	concept inclusion
FEMALE $\sqsubseteq$ PERSON	concept inclusion
PERSON $\sqsubseteq$ $\exists$hasFather	concept inclusion
$\exists$hasFather^{-} $\sqsubseteq$ MALE	concept inclusion
PERSON $\sqsubseteq$ $\exists$hasMother	concept inclusion
$\exists$hasMother^{-} $\sqsubseteq$ FEMALE	concept inclusion
MALE $\sqsubseteq \neg$FEMALE	concept disjointness
funct(hasMother)	role functionality

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$_A$ 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: $T \circ U$
- 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-Lite$_F$):

TBox: $\{ \exists \mathbf{WillPlay} \sqsubseteq \mathbf{AvailablePlayer}, \quad \mathbf{AvailablePlayer} \sqsubseteq \mathbf{Player},$
$\mathbf{Injured} \sqsubseteq \neg \mathbf{AvailablePlayer} \}$

ABox: $\{ \mathbf{WillPlay}(\mathbf{John}, \mathbf{Allstargame09}) \}$

The above ontology implies:

$\mathbf{AvailablePlayer}(\mathbf{John})$

$\mathbf{Player}(\mathbf{John})$

$\neg \mathbf{Injuried}(\mathbf{John})$

update: $\mathbf{Injuried}(\mathbf{John})$

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

$\{ \mathbf{Injuried}(\mathbf{John}), \ \mathbf{Player}(\mathbf{John}) \}$

Notice: John remains a player, and this would not be captured by simply removing $\mathbf{WillPlay}(\mathbf{John}, \mathbf{Allstargame09})$ from the ABox

Non-expressible updates over DL ontologies

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

TBox: $\{$ $\mathbf{ActivePlayer} \sqsubseteq \exists\mathbf{WillPlay}$, $\exists\mathbf{WillPlay}^- \sqsubseteq \mathbf{Game}$, $\exists\mathbf{WillPlay} \sqsubseteq \neg\, \mathbf{Injured}$ $\}$

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

The above ontology implies:

$\exists\mathbf{x}\,.\,\mathbf{WillPlay(John,x)} \wedge \mathbf{Game(x)}$

$\neg\mathbf{Injuried(John)}$

update: $\mathbf{Injuried(John)}$

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

(the updated ontology implies the sentence $\exists\mathbf{x}\,.\,\mathbf{Game(x)}$)

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 $L(x)$ **where** $q(x)$ (add information)
erase $L(x)$ **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:

$$\text{TELL}([\textbf{update } L(x) \textbf{ where } q(x)], s) = s \circ_T \bigcup_{t \in \text{ASK}(q(x),s)} L(t)$$
$$\text{if } Mod(T \cup \bigcup_{t \in \text{ASK}(q(x),s)} L(t)) \neq \emptyset$$

$$\text{TELL}([\textbf{erase } L(x) \textbf{ where } q(x)], s) = s \bullet_T \bigcup_{t \in \text{ASK}(q(x),s)} L(t)$$
$$\text{if } Mod(T \cup \neg \bigcup_{t \in \text{ASK}(q(x),s)} L(t)) \neq \emptyset$$

where we embedded the approximation in the semantics of update $\circ_T$ and erasure $\bullet_T$

Programs

We adopt a variant of Golog/ConGolog as formalism for programs: we concentrate on the deterministic fragment (extension to larger languages is easy):

a	action *(cf.* TELL*)*
ε	empty sequence of actions
$\delta 1; \delta 2$	sequential composition
if φ **then** $\delta 1$ **else** $\delta 2$	if-then-else
while φ **do** δ	while
pick $q(x).\delta[x]$	pick *(according to a given choice function)*

Transition semantics

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

- Given a program δ and a ontology s, **compute the ontology s' and the program δ' that remains to be executed after a single step a of δ in s.**

  Formally, define the **relation** $Trans$, denoted by " $\xrightarrow{a}$ ":

$$(\delta, s) \xrightarrow{a} (\delta', s')$$

- Assert when a program δ can be considered **successfully terminated** in an ontology s.

  Formally, define a **predicate** $Final$, denoted by " $\surd$":

$$(\delta, s)\surd$$

$Trans$ and $Final$ can be defined inductively in a standard way, using the so-called **transition (structural) rules** [Plotkin81, Nielson&Nielson99]

Structural rules

The structural rules have the following schema:

$$\frac{\text{CONSEQUENT}}{\text{ANTECEDENT}} \quad \text{if SIDE-CONDITION}$$

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
- ANTECEDENT, SIDE-CONDITION and 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

$$act: \quad \frac{(a, s) \xrightarrow{a} (\epsilon, \textsc{tell}(a, s))}{\textit{true}} \quad \text{if } a \text{ is executable in } s$$

$$seq: \quad \frac{(\delta_1; \delta_2,\ s) \xrightarrow{a} (\delta_1'; \delta_2, s')}{(\delta_1, s) \xrightarrow{a} (\delta_1'; s')} \qquad \frac{(\delta_1; \delta_2,\ s) \xrightarrow{a} (\delta_2', s')}{(\delta_2, s) \xrightarrow{a} (\delta_2'; s')} \quad \text{if } (\delta_1, s)\checkmark$$

$$if: \quad \frac{(\textbf{if } \phi \textbf{ then } \delta_1 \textbf{ else } \delta_2, s) \xrightarrow{a} (\delta_1', s')}{(\delta_1, s) \xrightarrow{a} (\delta_1', s')} \quad \text{if } \textsc{ask}(\phi, s) = \textit{true}$$

$$\frac{(\textbf{if } \phi \textbf{ then } \delta_1 \textbf{ else } \delta_2, s) \xrightarrow{a} (\delta_2', s')}{(\delta_2, s) \xrightarrow{a} (\delta_2', s')} \quad \text{if } \textsc{ask}(\phi, s) = \textit{false}$$

$$while: \quad \frac{(\textbf{while } \phi \textbf{ do } \delta, s) \xrightarrow{a} (\delta'; \textbf{while } \phi \textbf{ do } \delta, s')}{(\delta, s) \xrightarrow{a} (\delta', s')} \quad \text{if } \textsc{ask}(\phi, s) = \textit{true}$$

$$pick: \quad \frac{(\textbf{pick } q(\vec{x}).\ \delta[x], s) \xrightarrow{a} (\delta'[\vec{t}], s')}{(\delta[\vec{t}], s) \xrightarrow{a} (\delta'[\vec{t}], s')} \quad (\textit{for } \vec{t} = \textsc{choice}[\textsc{ask}(q(\vec{x}), s)])$$

Final rules for programs

$$\epsilon : \frac{(\epsilon, s)^{\checkmark}}{\textit{true}} \qquad seq : \frac{(\delta_1 ; \delta_2,\ s)^{\checkmark}}{(\delta_1, s)^{\checkmark} \ \wedge\ (\delta_2 ; s)^{\checkmark}}$$

$$if : \quad \frac{(\textbf{if } \phi \textbf{ then } \delta_1 \textbf{else } \delta_2, s)^{\checkmark}}{(\delta_1, s)^{\checkmark}} \ \text{if } \textsc{ask}(\phi, s) = \textit{true}$$

$$\frac{(\textbf{if } \phi \textbf{ then } \delta_1 \textbf{else } \delta_2, s)^{\checkmark}}{(\delta_2, s)^{\checkmark}} \ \text{if } \textsc{ask}(\phi, s) = \textit{false}$$

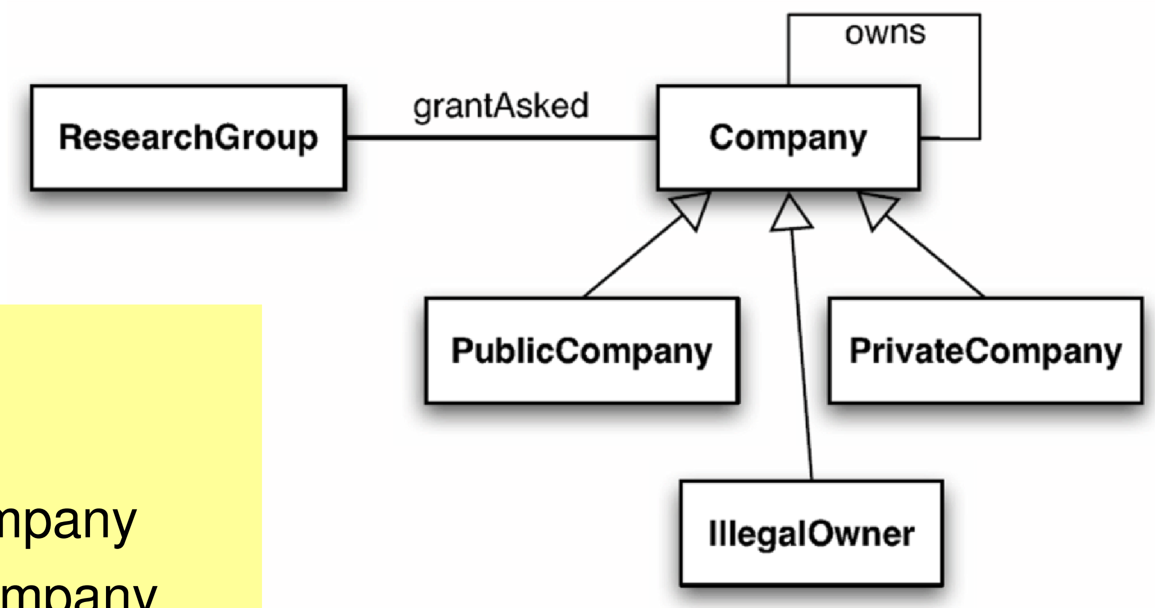
$$while : \quad \frac{(\textbf{while } \phi \textbf{ do } \delta, s)^{\checkmark}}{\textit{true}} \ \text{if } \textsc{ask}(\phi, s) = \textit{false}$$

$$\frac{(\textbf{while } \phi \textbf{ do } \delta, s)^{\checkmark}}{(\delta, s)^{\checkmark}} \ \text{if } \textsc{ask}(\phi, s) = \textit{true}$$

$$\textit{pick} : \quad \frac{(\textbf{pick } q(\vec{x}).\ \delta[\vec{x}], s)^{\checkmark}}{(\delta[\vec{t}], s)^{\checkmark}} \ \textit{(for } \vec{t} = \textsc{choice}[\textsc{ask}(q(\vec{x}), s)]\textit{)}$$

Example (cont.)

consider the following ontology on companies and grants:



$\exists owns \sqsubseteq Company$
$\exists owns^- \sqsubseteq Company$
$PublicCompany \sqsubseteq Company$
$PrivateCompany \sqsubseteq Company$
$\exists GrantAsked \sqsubseteq ResearchGroup$
$\exists GrantAsked^- \sqsubseteq Company$
$IllegalOwner \sqsubseteq Company$

Example (cont.)

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

(`temp` 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 $(\delta, s)\surd$ can be done in PTIME for *DL-Lite$_F$* ontologies (in (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 δ_0 over an ontology s_0 can be done in PTIME for *DL-Lite$_F$* ontologies (in (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 δ_0 over an ontology s_0, computing the **resulting program** δ_n **and ontology** s_n can be done in PTIME for *DL-Lite$_F$* ontologies (in (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 $\mathit{DL\text{-}Lite}_F$ 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