Answer Set Programming for the Semantic Web

Tutorial

Thomas Eiter, Roman Schindlauer (TU Wien)
Giovambattista Ianni (TU Wien, Univ. della Calabria)
Axel Polleres (Univ. Rey Juan Carlos, Madrid)

Supported by IST REWERSE, FWF Project P17212-N04, CICyT project TIC-2003-9001-C02.
Unit 5 – An ASP Extension: Nonmonotonic dl-Programs

T. Eiter

KBS Group, Institute of Information Systems, TU Vienna

European Semantic Web Conference 2006
Unit Outline

1. Introduction
2. dl-Programs
3. Answer Set Semantics
4. Applications and Properties
5. Further Aspects
Social Dinner Scenario (cont’d)

- Instead of a native, simple ontology inside the program, an external ontology should be used.
- An ontology is available, formulated in OWL, which contains information about available wine bottles, as instances of a concept *Wine*.
- It has further concepts *SweetWine*, *DryWine*, *RedWine* and *WhiteWine* for different types of wine.
• Instead of a native, simple ontology inside the program, an external ontology should be used.

• An ontology is available, formulated in OWL, which contains information about available wine bottles, as instances of a concept *Wine*.

• It has further concepts *SweetWine*, *DryWine*, *RedWine* and *WhiteWine* for different types of wine.
• Instead of a native, simple ontology inside the program, an external ontology should be used.

• An ontology is available, formulated in OWL, which contains information about available wine bottles, as instances of a concept *Wine*.

• It has further concepts *SweetWine*, *DryWine*, *RedWine* and *WhiteWine* for different types of wine.
Instead of a native, simple ontology inside the program, an external ontology should be used.

An ontology is available, formulated in OWL, which contains information about available wine bottles, as instances of a concept `Wine`.

It has further concepts `SweetWine`, `DryWine`, `RedWine` and `WhiteWine` for different types of wine.

How to use this ontology from the logic program?

How to ascribe a semantics for this usage?
Nonmonotonic Description Logic Programs

- An extension of answer set programs with *queries to DL knowledge bases* (through *dl-atoms*)
- Formal semantics for emerging programs (*nonmonotonic dl-programs*), fostering the *interfacing view*
  ⇒ Clean technical separation of DL engine and ASP solver
- New generalized definitions of answer sets of a general dl-program

Important: *bidirectional flow of information*
⇒ The logic program also may provide *input to DL knowledge base*

Prototype implementation, examples
http://www.kr.tuwien.ac.at/staff/roman/semweblp/
Nonmonotonic Description Logic Programs

- An extension of answer set programs with queries to DL knowledge bases (through \textit{dl-atoms})
- Formal semantics for emerging programs (\textit{nonmonotonic dl-programs}), fostering the \textit{interfacing view} \\
  ⇒ Clean technical separation of DL engine and ASP solver
- New generalized definitions of answer sets of a general dl-program

Important: \textit{bidirectional flow of information}

⇒ The logic program also may provide input to DL knowledge base

Prototype implementation, examples

http://www.kr.tuwien.ac.at/staff/roman/semweblp/
Nonmonotonic Description Logic Programs

- An extension of answer set programs with *queries to DL knowledge bases* (through *dl-atoms*)
- Formal semantics for emerging programs (*nonmonotonic dl-programs*), fostering the *interfacing view*
  ⇒ Clean technical separation of DL engine and ASP solver
- New generalized definitions of answer sets of a general dl-program

**Important**: *bidirectional flow of information*

⇒ The logic program also may provide *input to DL knowledge base*

**Prototype implementation, examples**

http://www.kr.tuwien.ac.at/staff/roman/semweblp/
dl-Atoms

Approach to enable a call to a DL engine in ASP:

- Pose a query, $Q$, to a DL knowledge base, $L$
- Allow to modify the extensional part (ABox) of $KB$
- Query evaluates to true, iff $Q$ is provable in modified $L$. 

Examples:

- DL\[Wine\](“ChiantiClassico”)
- DL\[Wine\](X)
- DL\[DryWine\] ⊎ my\_dry; Wine\(\)(W)

Add all assertions \(DryWine\left(c\right)\) to the ABox (extensional part) of $L$, such that $my\_dry\left(c\right)$ holds.
Approach to enable a call to a DL engine in ASP:

- Pose a query, $Q$, to a DL knowledge base, $L$
- Allow to modify the extensional part (ABox) of $KB$
- Query evaluates to true, iff $Q$ is provable in modified $L$.

**Examples: wine ontology**

- $DL[\text{Wine}](\text{“ChiantiClassico”})$
- $DL[\text{Wine}](X)$
- $DL[\text{DryWine} \sqcup my\_dry; \text{Wine}](W)$

  add all assertions $\text{DryWine}(c)$ to the ABox (extensional part) of $L$, such that $my\_dry(c)$ holds.
dl-Atoms

Approach to enable a call to a DL engine in ASP:

- Pose a query, $Q$, to a DL knowledge base, $L$
- Allow to modify the extensional part (ABox) of $KB$
- Query evaluates to true, iff $Q$ is provable in modified $L$.

**Examples:** wine ontology

- $DL[\text{Wine}](\text{“ChiantiClassico”})$
- $DL[\text{Wine}](X)$
- $DL[\text{DryWine} \sqcup \text{my\_dry}; \text{Wine}](W)$
  
  add all assertions $\text{DryWine}(c)$ to the ABox (extensional part) of $L$, such that $\text{my\_dry}(c)$ holds.
dl-Atoms

Approach to enable a call to a DL engine in ASP:

- Pose a query, $Q$, to a DL knowledge base, $L$
- Allow to modify the extensional part (ABox) of $KB$
- Query evaluates to true, iff $Q$ is provable in modified $L$.

**Examples: wine ontology**

- $DL[\text{Wine}](\text{"ChiantiClassico"})$
- $DL[\text{Wine}](X)$
- $DL[\text{DryWine} \sqcup my\_dry; \text{Wine}](W)$

  add all assertions $\text{DryWine}(c)$ to the ABox (extensional part) of $L$, such that $my\_dry(c)$ holds.
A **dl-atom** has the form

\[ DL[S_1 op_1 p_1, \ldots, S_m op_m p_m; Q](t), \quad m \geq 0, \]

where

- each \( S_i \) is either a concept or a role
- \( op_i \in \{\cup, \cup\} \)
- \( p_i \) is a unary resp. binary predicate (**input predicate**),
- \( Q(t) \) is a **DL query**.

**Intuitively:**

- \( op_i = \cup \) increases \( S_i \) by \( p_i \).
- \( op_i = \cup \) increases \( \neg S_i \) by \( p_i \).
**A dl-atom has the form**

\[ DL[S_1 \text{op}_1 p_1, \ldots, S_m \text{op}_m p_m; Q](t), \quad m \geq 0, \]

where

- each \( S_i \) is either a concept or a role
- \( \text{op}_i \in \{\uplus, \uplus\} \)
- \( p_i \) is a unary resp. binary predicate (input predicate),
- \( Q(t) \) is a DL query.

**Intuitively:**

- \( \text{op}_i = \uplus \) increases \( S_i \) by \( p_i \).
- \( \text{op}_i = \uplus \) increases \( \neg S_i \) by \( p_i \).
A dl-atom has the form

\[ DL[S_1 o_1 p_1, \ldots, S_m o_m p_m; Q](t), \quad m \geq 0, \]

where

- each \( S_i \) is either a concept or a role
- \( o_i \in \{\cup, \sqcup\} \),
- \( p_i \) is a unary resp. binary predicate (input predicate),
- \( Q(t) \) is a DL query.

Intuitively:

- \( o_i = \cup \) increases \( S_i \) by \( p_i \).
- \( o_i = \sqcup \) increases \( \neg S_i \) by \( p_i \).
A `dl-atom` has the form

\[
DL[S_1 op_1 p_1, \ldots, S_m op_m p_m; Q](t), \quad m \geq 0,
\]

where

- each \( S_i \) is either a concept or a role
- \( op_i \in \{\cup, \subseteq\} \)
- \( p_i \) is a unary resp. binary predicate \((input\ predicate)\)
- \( Q(t) \) is a `DL` query.

**Intuitively:**

\[
\begin{align*}
op_i = \cup & \quad \text{increases } S_i \text{ by } p_i. \\
op_i = \subseteq & \quad \text{increases } \neg S_i \text{ by } p_i.
\end{align*}
\]
A \textit{dl-atom} has the form

\[ DL[S_1 \, op_1 \, p_1, \ldots, S_m \, op_m \, p_m; \, Q](t), \quad m \geq 0, \]

where

- each \( S_i \) is either a concept or a role
- \( op_i \in \{\cup, \sqcup\} \),
- \( p_i \) is a unary resp. binary predicate (\textit{input predicate}),
- \( Q(t) \) is a \textit{DL query}.

\textbf{Intuitively:}

- \( op_i = \cup \) increases \( S_i \) by \( p_i \).
- \( op_i = \sqcup \) increases \( \neg S_i \) by \( p_i \).
A DL query $Q(t)$ is one of

(a) a concept inclusion axiom $C \sqsubseteq D$, or its negation $\neg(C \sqsubseteq D)$,
(b) $C(t)$ or $\neg C(t)$, for a concept $C$ and term $t$, or
(c) $R(t_1, t_2)$ or $\neg R(t_1, t_2)$, for a role $R$ and terms $t_1, t_2$.

Remarks:

- Further queries are conceivable (e.g., conjunctive queries)
- The queries above are standard queries.
A \textit{dl-rule} \( r \) is of form

\[ a \leftarrow b_1, \ldots, b_k, \text{not } b_{k+1}, \ldots, \text{not } b_m, \quad m \geq k \geq 0, \]

where

- \( a \) is a classical first-order literal
- \( b_1, \ldots, b_m \) are classical first-order literals or \( \text{dl-} \)atoms (no function symbols).

\textbf{Definition}

A \textit{nonmonotonic description logic (dl-)} program \( KB = (L, P) \) consists of

- a knowledge base \( L \) in a description logic (\( \bigcup * \Box \)),
- a finite set of \( \text{dl-} \)rules \( P \).
dl-Programs

A dl-rule $r$ is of form

$$a \leftarrow b_1, \ldots, b_k, \text{not } b_{k+1}, \ldots, \text{not } b_m, \quad m \geq k \geq 0,$$

where

- $a$ is a classical first-order literal
- $b_1, \ldots, b_m$ are classical first-order literals or dl-atoms (no function symbols).

**Definition**

A nonmonotonic description logic (dl-) program $KB = (L, P)$ consists of

- a knowledge base $L$ in a description logic ($\bigcup *\text{Box}$),
- a finite set of dl-rules $P$. 
Task

*Modify* `wineCover09a.dlp` *by fetching the wines now from the ontology.*

For instance:

```
wineBottle(X) :- DL["Wine"](X).
```

Fetches all the known instances of *Wine*.

Think at how the “*isA*” predicate could be redefined in terms of *dl-atoms*:

```
isA(X,"SweetWine") :- ?
isA(X,"DessertWine") :- ?
isA(X,"ItalianWine") :- ?
```

Solution at
Task

Modify *wineCover09a.dlp* by fetching the wines now from the ontology.

For instance:

```
wineBottle(X) :- DL["Wine"](X).
```

Fetches all the known instances of *Wine*.

Think at how the “isA” predicate could be redefined in terms of dl-atoms

```
isA(X,“SweetWine”) :- DL[SweetWine](X).
isA(X,“DessertWine”) :- DL[DessertWine](X).
isA(X,“ItalianWine”) :- DL[ItalianWine](X).
```

Solution at *wineCover9b.dlp*
Social Dinner X

- Suppose now that we learn that there is a bottle, “SelaksIceWine”, which is a white wine and not dry.
- We may add this information to the logic program by facts\(^1\):

  \[
  \text{white(“SelaksIceWine”). not\_dry(“SelaksIceWine”).}
  \]

- In our program, we may pass this information to the ontology by adding in the \(\text{dl}\)-atoms the modification

  \[
  \text{WhiteWine} \cup \text{white, DryWine} \cup \text{not\_dry}.
  \]

E.g., \(\text{DL[Wine]}(X)\) is changed to

\[
\text{DL[WhiteWine +\_ white, DryWine -\_ not\_dry; Wine]}(X).
\]

\(^1\)See \text{wineCover09c.dlp}
Semantics of $KB = (L, P)$

- $HB_P^\Phi$: Set of all ground (classical) literals with predicate symbol in $P$ and constants from finite relational alphabet $\Phi$.
- Constants: those in $P$ and (all) individuals in the ABox of $L$.
- Herbrand interpretation: consistent subset $I \subseteq HB_P^\Phi$
  - $I \models_L \ell$ for classical ground literal $\ell$, iff $\ell \in I$;
  - $I \models_L DL[S_1 op_1 p_1 \ldots, S_m op_m p_m; Q](c)$ if and only if
    $L \cup A_1(I) \cup \cdots \cup A_m(I) \models Q(c)$,
    where
    - $A_i(I) = \{S_i(e) | p_i(e) \in I\}$, for $op_i = \lor$;
    - $A_i(I) = \{\neg S_i(e) | p_i(e) \in I\}$, for $op_i = \lor$.
- The models of $KB = (L, P)$ are the joint models of all rules in $P$ (defined as usual).
Semantics of $KB = (L, P)$

- $HB_P^\Phi$: Set of all ground (classical) literals with predicate symbol in $P$ and constants from finite relational alphabet $\Phi$.
- Constants: those in $P$ and (all) individuals in the ABox of $L$.
- Herbrand interpretation: consistent subset $I \subseteq HB_P^\Phi$
  
  - $I \models_L \ell$ for classical ground literal $\ell$, iff $\ell \in I$;
  
  - $I \models_L DL[S_1 op_1 p_1 \ldots, S_m op_m p_m; Q](c)$ if and only if
    
    \[
    L \cup A_1(I) \cup \cdots \cup A_m(I) \models Q(c),
    \]
    
    where
    
    - $A_i(I) = \{ S_i(e) | p_i(e) \in I \}$, for $op_i = \cup$;
    - $A_i(I) = \{ \neg S_i(e) | p_i(e) \in I \}$, for $op_i = \cup$.
  
  - The models of $KB = (L, P)$ are the joint models of all rules in $P$ (defined as usual)
Semantics of $KB = (L, P)$

- $HB_P^\Phi$: Set of all ground (classical) literals with predicate symbol in $P$ and constants from finite relational alphabet $\Phi$.
- Constants: those in $P$ and (all) individuals in the ABox of $L$.
- Herbrand interpretation: consistent subset $I \subseteq HB_P^\Phi$
  - $I \models_L \ell$ for classical ground literal $\ell$, iff $\ell \in I$;
  - $I \models_L DL[S_1 op_1 p_1 \ldots, S_m op_m p_m; Q](c)$ if and only if
    \[ L \cup A_1(I) \cup \cdots \cup A_m(I) \models Q(c), \]
    where
    - $A_i(I) = \{S_i(e) \mid p_i(e) \in I\}$, for $op_i = \cup$;
    - $A_i(I) = \{\neg S_i(e) \mid p_i(e) \in I\}$, for $op_i = \cup$.
- The models of $KB = (L, P)$ are the joint models of all rules in $P$ (defined as usual).
Semantics of $KB = (L, P)$

- $HB_P^\Phi$: Set of all ground (classical) literals with predicate symbol in $P$ and constants from finite relational alphabet $\Phi$.
- Constants: those in $P$ and (all) individuals in the ABox of $L$.
- Herbrand interpretation: consistent subset $I \subseteq HB_P^\Phi$
  - $I \models_L \ell$ for classical ground literal $\ell$, iff $\ell \in I$;
  - $I \models_L DL[S_1 op_1 p_1 \ldots, S_m op_m p_m; Q](c)$ if and only if
    $$L \cup A_1(I) \cup \cdots \cup A_m(I) \models Q(c),$$
    where
    - $A_i(I) = \{S_i(e) \mid p_i(e) \in I\}$, for $op_i = \cup$;
    - $A_i(I) = \{-S_i(e) \mid p_i(e) \in I\}$, for $op_i = \cup$.
- The models of $KB = (L, P)$ are the joint models of all rules in $P$
  (defined as usual)
Examples

• Suppose $L \models Wine(“TaylorPort”)$, and $I$ contains $wineBottle(“TaylorPort”)$. Then $I \models_L DL[“Wine”](“TaylorPort”) \land I \models_L wineBottle(“TaylorPort”) : - DL[“Wine”](“TaylorPort”)

• Suppose $I = \{white(“siw”), not\_dry(“siw”)\}$. Then $I \models_L DL[“WhiteWine” \uplus white, “DryWine” \uplus not\_dry; “Wine”](“siw”)$. 
Examples

- Suppose \( L \models Wine(“TaylorPort”) \), and \( I \) contains \( wineBottle(“TaylorPort”) \)

Then \( I \models L \ DL[“Wine”](“TaylorPort”) \) and

\[
I \models L \ wineBottle(“TaylorPort”) \iff DL[“Wine”](“TaylorPort”)
\]

- Suppose \( I = \{ \text{white(“siw”), not\_\_dry(“siw”)} \} \).

Then \( I \models L \ DL[“WhiteWine” \equiv \text{white, “DryWine”}\equiv \text{not\_\_dry}; “Wine”](“siw”) \)
• Suppose $L \not\models DL["Wine"]("Milk")$. Then for every $I$,

$I \models_L \text{compliant}(joe,"Milk") \iff DL["Wine"]("Milk")$

$I \models_L \text{not } DL["Wine"]("Milk")$.

• Note that $I \models_L \text{not } DL["Wine"]("Milk")$ is different from

$I \models_L DL[\neg "Wine"]("Milk")$.

• Inconsistency of $L$ is revealed with unsatisfiable DL queries:

$\text{inconsistent} :\iff DL["Wine" \sqsubseteq \neg "Wine"]$

Shorthand: $DL[\bot]$

• Consistency can be checked by

$\text{consistent} :\iff \text{not } DL["Wine" \sqsubseteq \neg "Wine"]$
Examples /2

- Suppose $L \not\models DL["Wine"]("Milk").$ Then for every $I,$
  
  $I \models_L \text{compliant}(joe,"Milk") \iff DL["Wine"]("Milk")$
  
  $I \models_L \text{not } DL["Wine"]("Milk").$

- Note that $I \models_L \text{not } DL["Wine"]("Milk")$ is different from
  
  $I \models_L DL[\neg "Wine"]("Milk").$

- Inconsistency of $L$ is revealed with unsatisfiable DL queries:
  
  $\text{inconsistent} :\iff DL["Wine" \subseteq \neg "Wine"]$

  Shorthand: $DL[\bot]$

- Consistency can be checked by
  
  $\text{consistent} :\iff \text{not } DL["Wine" \subseteq \neg "Wine"]$
Examples /2

• Suppose $L \not\models DL["Wine"]("Milk").$ Then for every $I,$

$$I \models_L\text{ compliant}(joe,"Milk") \leftarrow DL["Wine"]("Milk")$$

$$I \models_L\text{ not } DL["Wine"]("Milk").$$

• Note that $I \models_L\text{ not } DL["Wine"]("Milk")$ is different from

$I \models_L DL[\neg "Wine"]("Milk").$

• Inconsistency of $L$ is revealed with unsatisfiable DL queries:

$$\text{inconsistent} :- DL["Wine" \sqsubseteq \neg "Wine"]$$

Shorthand: $DL[\bot]$

• Consistency can be checked by

$$\text{consistent} :- \text{not } DL["Wine" \sqsubseteq \neg "Wine"]$$
Examples /2

- Suppose $L \not\models DL[“Wine”](“Milk”). Then for every $I$,
  
  \[ I \models_L \text{compliant}(\text{joe, “Milk”}) :- DL[“Wine”](“Milk”) \]
  
  \[ I \models_L \text{not } DL[“Wine”](“Milk”) \]

- Note that $I \models_L \text{not } DL[“Wine”](“Milk”) \text{ is different from}$
  
  \[ I \models_L DL[\neg “Wine”](“Milk”) \]

- Inconsistency of $L$ is revealed with unsatisfiable DL queries:

  \[ \text{inconsistent} :- DL[“Wine” \sqsubseteq \neg “Wine”] \]

  Shorthand: $DL[\bot]$

- Consistency can be checked by

  \[ \text{consistent} :- \text{not } DL[“Wine” \sqsubseteq \neg “Wine”] \]
Answer Sets of positive $KB = (L, P)$ (no $not$ in $P$):

- $KB = (L, P)$ has the least model $lm(KB)$ (if satisfiable)
- The single answer set of $KB$ is $lm(KB)$

Answer Sets of general $KB = (L, P)$:

- Use a reduct $KB^I$ akin to the Gelfond-Lifschitz (GL) reduct:
  \[ KB^I = (L, P^I) \]

  where $P^I$ is the GL-reduct of $P$ wrt. $I$ (treat $dl$-atoms like regular atoms)
- $I$ is an answer set of $KB$ iff $I = lm(KB^I)$. 
Answer Sets

Answer Sets of positive $KB = (L, P)$ (no not in $P$):

- $KB = (L, P)$ has the least model $lm(KB)$ (if satisfiable)
- The single answer set of $KB$ is $lm(KB)$

Answer Sets of general $KB = (L, P)$:

- Use a reduct $KB^I$ akin to the Gelfond-Lifschitz (GL) reduct:
  $$KB^I = (L, P^I)$$

  where $P^I$ is the GL-reduct of $P$ wrt. $I$ (treat dl-atoms like regular atoms)

- $I$ is an answer set of $KB$ iff $I = lm(KB^I)$. 
Some Semantical Properties

- **Existence**: Positive dl-programs without “¬” and constraints always have an answer set.

- **Uniqueness**: Layered use of “not” (stratified dl-program) ⇒ single answer set.

- **Conservative extension**: For dl-program $KB = (L, P)$ without dl-atoms, the answer sets are the answer sets of $P$.

- **Minimality**: answer sets of $KB$ are models, and moreover minimal models.

- **Fixpoint Semantics**: Positive and stratified dl-programs with monotone dl-atoms possess fixpoint characterizations of the answer set.
Some Semantical Properties

- **Existence**: Positive dl-programs without “¬” and constraints always have an answer set.

- **Uniqueness**: Layered use of “not” (stratified dl-program) ⇒ single answer set.

- **Conservative extension**: For dl-program $KB = (L, P)$ without dl-atoms, the answer sets are the answer sets of $P$.

- **Minimality**: answer sets of $KB$ are models, and moreover minimal models.

- **Fixpoint Semantics**: Positive and stratified dl-programs with monotone dl-atoms possess fixpoint characterizations of the answer set.
Some Semantical Properties

- **Existence**: Positive dl-programs without "¬" and constraints always have an answer set.
- **Uniqueness**: Layered use of "not" (stratified dl-program) ⇒ single answer set.
- **Conservative extension**: For dl-program $KB = (L, P)$ without dl-atoms, the answer sets are the answer sets of $P$.
- **Minimality**: answer sets of $KB$ are models, and moreover minimal models.
- **Fixpoint Semantics**: Positive and stratified dl-programs with monotone dl-atoms possess fixpoint characterizations of the answer set.
Some Semantical Properties

- **Existence**: Positive dl-programs without “¬” and constraints always have an answer set.

- **Uniqueness**: Layered use of “not” (stratified dl-program) ⇒ single answer set.

- **Conservative extension**: For dl-program $KB = (L, P)$ without dl-atoms, the answer sets are the answer sets of $P$.

- **Minimality**: answer sets of $KB$ are models, and moreover minimal models.

- **Fixpoint Semantics**: Positive and stratified dl-programs with monotone dl-atoms possess fixpoint characterizations of the answer set.
Some Semantical Properties

- **Existence**: Positive dl-programs without “¬” and constraints always have an answer set.
- **Uniqueness**: Layered use of “not” (stratified dl-program) ⇒ single answer set.
- **Conservative extension**: For dl-program \( KB = (L, P) \) without dl-atoms, the answer sets are the answer sets of \( P \).
- **Minimality**: answer sets of \( KB \) are models, and moreover minimal models.
- **Fixpoint Semantics**: Positive and stratified dl-programs with monotone dl-atoms possess fixpoint characterizations of the answer set.
Some Reasoning Applications

- *dl*-atoms allow to query description knowledge base repeatedly
- We might use *dl*-programs as rule-based “glue” for inferences on a DL base.
- In this way, inferences can be combined
- Here, we show some applications where non-monotonic and minimization features of *dl*-programs can be exploited
Some Reasoning Applications

- $\textit{dl}$-atoms allow to query description knowledge base repeatedly

- We might use $\textit{dl}$-programs as rule-based "glue" for inferences on a DL base.

- In this way, inferences can be combined

- Here, we show some applications where non-monotonic and minimization features of $\textit{dl}$-programs can be exploited
Some Reasoning Applications

- dl-atoms allow to query description knowledge base repeatedly
- We might use dl-programs as rule-based “glue” for inferences on a DL base.
- In this way, inferences can be combined
- Here, we show some applications where non-monotonic and minimization features of dl-programs can be exploited
• dl-atoms allow to query description knowledge base repeatedly

• We might use dl-programs as rule-based “glue” for inferences on a DL base.

• In this way, inferences can be combined

• Here, we show some applications where non-monotonic and minimization features of dl-programs can be exploited
Closed World Assumption (CWA)

Reiter’s Closed World Assumption (CWA)

For ground atom \( p(c) \), infer \( \neg p(c) \) if \( KB \notmodels p(c) \)

- Express CWA for concepts \( C_1, \ldots, C_k \) wrt. individuals in \( L \):
  
  \[
  \neg c_1(X) \leftarrow \text{not} \ DL[C_1](X) \\
  \ldots \\
  \neg c_k(X) \leftarrow \text{not} \ DL[C_k](X)
  \]

- CWA for roles \( R \): easy extension
Query Answering under CWA

**Example:** \( L = \{ \text{SparklingWine}(\text{“VeuveCliquot”}), \text{Sparklingwine} \sqcap \neg \text{WhiteWine})(\text{“Lambrusco”}) \} \).

**Query:** \( \text{WhiteWine}(\text{“VeuveCliquot”}) \) (Y/N)?
Query Answering under CWA

Example: \( L = \{ \text{SparklingWine("VeuveCliquot")}, \) \\
\( \quad (\text{Sparklingwine} \sqcap \neg \text{WhiteWine})("\text{Lambrusco}") \} \).

Query: \( \text{WhiteWine("VeuveCliquot") (Y/N)}? \)

Add CWA-literals to \( L \):

\[
\bar{sp}(X) \leftarrow \neg DL[\text{SparklingWine}](X) \\
\bar{ww}(X) \leftarrow \neg DL[\text{WhiteWine}](X) \\
ww(X) \leftarrow DL[\text{SparklingWine} \cup \bar{sp}, \\
\text{WhiteWine} \cup \bar{ww}; \text{WhiteWine}](X)
\]

Ask whether \( KB \models ww("VeuveCliquot") \) or \( KB \models \bar{ww}("VeuveCliquot") \)
Extended CWA

- CWA can be inconsistent (disjunctive knowledge)

- Example:
  Knowledge base

  \[ L = \{ \text{Artist(“Jody”), Artist } \equiv \text{Painter } \sqcap \text{Singer} \} \]

- CWA for Painter, Singer adds

  \[ \neg \text{Painter(“Jody”), } \neg \text{Singer(“Jody”).} \]

- This implies \[ \neg \text{Artist(“Jody”)}. \]
Extended CWA

- CWA can be inconsistent (disjunctive knowledge)

- Example:
  Knowledge base

  \[ L = \{ \text{Artist(“Jody”)}, \text{Artist} \equiv \text{Painter} \sqcup \text{Singer} \} \]

- CWA for Painter, Singer adds

  \[ \neg \text{Painter(“Jody”)}, \neg \text{Singer(“Jody”).} \]

- This implies \( \neg \text{Artist(“Jody”)} \)
ECWA singles out “minimal” models of $L$ wrt Painter and Singer (UNA in $L$ on ABox):

$$\bar{p}(X) \leftarrow \text{not } p(X)$$
$$\bar{s}(X) \leftarrow \text{not } s(X)$$
$$p(X) \leftarrow DL[\text{Painter} \cup \bar{p}, \text{Singer} \cup \bar{s}; \text{Painter}](X)$$
$$s(X) \leftarrow DL[\text{Painter} \cup \bar{p}, \text{Singer} \cup \bar{s}; \text{Singer}](X)$$

Answer sets:

$$M_1 = \{p(“Jody”), \bar{s}(“Jody”)\},$$
$$M_2 = \{s(“Jody”), \bar{p}(“Jody”)\}$$

Extendible to keep concepts “fixed”

$$\sim \text{ ECWA}(\phi; P; Q; Z)$$
ECWA singles out “minimal” models of $L$ wrt $Painter$ and $Singer$ (UNA in $L$ on ABox):

\[
\begin{align*}
\overline{p}(X) & \leftarrow \text{not } p(X) \\
\overline{s}(X) & \leftarrow \text{not } s(X) \\
p(X) & \leftarrow DL[\text{Painter}$;$\overline{p}$, $Singer$;$\overline{s}$; $Painter](X) \\
s(X) & \leftarrow DL[\text{Painter}$;$\overline{p}$, $Singer$;$\overline{s}$; $Singer](X)
\end{align*}
\]

Answer sets:

\[
\begin{align*}
M_1 & = \{p(“Jody”), \overline{s}(“Jody”)\}, \\
M_2 & = \{s(“Jody”), \overline{p}(“Jody”)\}
\end{align*}
\]

- Extendible to keep concepts “fixed”

$\leadsto$ ECWA($\phi$; $P$; $Q$; $Z$)
ECWA singles out “minimal” models of $L$ wrt $Painter$ and $Singer$ (UNA in $L$ on ABox):

$$\overline{p}(X) \leftarrow \text{not } p(X)$$

$$\overline{s}(X) \leftarrow \text{not } s(X)$$

$$p(X) \leftarrow DL[\text{Painter}\cup \overline{p}, \text{Singer}\cup \overline{s}; \text{Painter}](X)$$

$$s(X) \leftarrow DL[\text{Painter}\cup \overline{p}, \text{Singer}\cup \overline{s}; \text{Singer}](X)$$

Answer sets:

$$M_1 = \{p(“Jody”), \overline{s}(“Jody”)\},$$

$$M_2 = \{s(“Jody”), \overline{p}(“Jody”)\}$$

Extendible to keep concepts “fixed”

$$\sim ECWA(\phi; P; Q; Z)$$
Default Reasoning

Add simple default rules a la Poole (1988) on top of ontologies

**Example:** wine ontology

\[
L = \{ \text{SparklingWine}(“VeuveCliquot”),
(“SparklingWine” \sqcap \neg “WhiteWine”)(“Lambrusco”) \},
\]

Use default rule: Sparkling wines are white by default

\begin{align*}
r_1 & : \quad \text{white}(W) \leftarrow \text{DL}[\text{SparklingWine}](W), \neg \text{white}(W) \\
r_2 & : \quad \neg \text{white}(W) \leftarrow \text{DL}[\text{WhiteWine} \uplus \text{white}; \neg \text{WhiteWine}](W) \\
r_3 & : \quad f \leftarrow \neg f, \text{DL}[\bot] \quad /* \text{kill model if } L \text{ is inconsistent } */
\end{align*}

- In answer set semantics, \( r_2 \) effects maximal application of \( r_1 \).
- Answer Set: \( M = \{ \text{white(“VeuveCliquot”), } \neg \text{white(“Lambrusco”)}\} \)
Default Reasoning

Add simple default rules a la Poole (1988) on top of ontologies

**Example:** wine ontology

\[
L = \{ \text{SparklingWine(“VeuveCliquot”)}, \\
\text{ (“SparklingWine” ⊓¬ “WhiteWine”) (”Lambrusco”)} \},
\]

Use default rule: Sparkling wines are white by default

\[
r1 : \text{ white}(W) \leftarrow \text{DL[SparklingWine]}(W), \text{ not } \neg \text{white}(W) \\
r2 : \neg \text{white}(W) \leftarrow \text{DL[WhiteWine ⊔ white; } \neg \text{WhiteWine]}(W) \\
r3 : \text{ f } \leftarrow \text{not } \text{f}, \text{DL[⊥]} \quad /* \text{kill model if } L \text{ is inconsistent */}
\]

- In answer set semantics, \( r2 \) effects maximal application of \( r1 \).
- Answer Set: \( M = \{ \text{white(“VeuveCliquot”), } \neg \text{white(“Lambrusco”)}) \} \)
Further Aspects of dl-programs

- **Stratified dl-programs**: intuitively, composed of hierarchic layers of positive dl-programs linked via default negation. This generalization of the classic notion of stratification embodies a fragment of the language having single answer sets.

- Non-monotonic dl-atoms: Operator \(\bigcap\)
  
  \[ DL[WhiteWine \bigcap my\_WhiteWine](X) \]
  
  Constrain \(WhiteWine\) to \(my\_WhiteWine\)

- **Weak answer-set semantics** (Here: Strong answer sets)
  Treat also positive dl-atoms like \(not\)-literals in the reduct

- **Well-founded semantics**
  Generalization of the traditional well-founded semantics for normal logic programs.
Further Aspects of \( \text{dl} \)-programs

• **Stratified \( \text{dl} \)-programs**: intuitively, composed of hierarchic layers of positive \( \text{dl} \)-programs linked via default negation. This generalization of the classic notion of stratification embodies a fragment of the language having single answer sets.

• **Non-monotonic \( \text{dl} \)-atoms**: Operator \( \sqcap \)

\[
DL[\text{White Wine} \sqcap \text{my\_White Wine}](X)
\]

Constrain \( \text{White Wine} \) to \( \text{my\_White Wine} \)

• **Weak answer-set semantics** (Here: Strong answer sets)
  Treat also positive \( \text{dl} \)-atoms like \( \text{not} \)-literals in the reduct

• **Well-founded semantics**
  Generalization of the traditional well-founded semantics for normal logic programs.
Further Aspects of \( \text{dl} \)-programs

- Stratified \( \text{dl} \)-programs: intuitively, composed of hierarchic layers of positive \( \text{dl} \)-programs linked via default negation. This generalization of the classic notion of stratification embodies a fragment of the language having single answer sets.

- Non-monotonic \( \text{dl} \)-atoms: Operator \( \sqcap \)

\[
DL[\text{White Wine} \sqcap \text{my White Wine}](X)
\]

Constrain \( \text{White Wine} \) to \( \text{my White Wine} \)

- \textit{Weak answer-set semantics} (Here: Strong answer sets)
  Treat also positive \( \text{dl} \)-atoms like \( \text{not} \)- literals in the reduct

- \textit{Well-founded semantics}
  Generalization of the traditional well-founded semantics for normal logic programs.
Further Aspects of \(dl\)-programs

- **Stratified \(dl\)-programs**: intuitively, composed of hierarchic layers of positive \(dl\)-programs linked via default negation. This generalization of the classic notion of stratification embodies a fragment of the language having single answer sets.

- **Non-monotonic \(dl\)-atoms**: Operator \(\sqcap\)

\[
DL[\text{WhiteWine} \sqcap \text{my\_WhiteWine}](X)
\]

Constrain \(\text{WhiteWine}\) to \(\text{my\_WhiteWine}\)

- **Weak answer-set semantics** (Here: Strong answer sets)
  Treat also positive \(dl\)-atoms like \(\text{not}\)-literals in the reduct

- **Well-founded semantics**
  Generalization of the traditional well-founded semantics for normal logic programs.
Deciding strong answer set existence for dl-programs (completeness results)

<table>
<thead>
<tr>
<th>$KB = (L, P)$</th>
<th>$L$ in $SHIF(D)$</th>
<th>$L$ in $SHOIN(D)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>positive</td>
<td>EXP</td>
<td>NEXP</td>
</tr>
<tr>
<td>stratified</td>
<td>EXP</td>
<td>$P^{NEXP}$</td>
</tr>
<tr>
<td>general</td>
<td>$NEXP$</td>
<td>$NP^{NEXP}$</td>
</tr>
</tbody>
</table>

Recall: Satisfiability problem in

- $SHIF(D) / SHOIN(D)$ is EXP-/NEXP-complete (unary numbers).
- ASP is EXP-complete for positive/stratified programs $P$, and NEXP-complete for arbitrary $P$

**Key observation:** The number of ground dl-atoms is polynomial

- $NP^{NEXP} = P^{NEXP}$ is less powerful than disjunctive ASP ($\equiv NEXP^{NP}$)
- Similar results for query answering
NLP-DL Prototype

- Fully operational prototype: NLP-DL
  
  http://www.kr.tuwien.ac.at/staff/roman/semweblp/.

- Accepts ontologies formulated in OWL-DL (as processed by RACER) and a set of dl-rules, where ←, ∪, and ∪, are written as "::-", "+=", and "-=", respectively.

- Model computation: compute
  - the answer sets
  - the well-founded model

  Preliminary computation of the well-founded model may be exploited for optimization.

- Reasoning: both brave and cautious reasoning; well-founded inferences
Example: Review Assignment

It is given an ontology about scientific publications

- Concept *Author* stores authors
- Concept *Senior* (senior author)
- Concept *Club100* (authors with more than 100 paper)
- ...

- Goal: Assign submitted papers to reviewers
- Note: Precise definitions are not so important (encapsulation)
Review Assignment /2

Facts:

paper(subm1). author(subm1,"jdbr"). author(subm1,"htom").

paper(subm2). author(subm2,"teit"). author(subm2,"gian").

author(subm2,"rsch"). author(subm2,"apol").

The program committee:

pc("vlif"). pc("mgel"). pc("dfen"). pc("fley"). pc("smil").

pc("mkif"). pc("ptra"). pc("ggot"). pc("ihor").

All PC members are in the “Club100” with more than 100 papers:

Consider all senior researchers as candidate reviewers adding the club100 information to the OWL knowledge base:

cand(X,P) :- paper(P), DL["club100" += pc;"senior"](X).
Review Assignment /2

Facts:

\[
\begin{align*}
\text{paper(subm1). author(subm1,"jdbr"). author(subm1,"htom").} \\
\text{paper(subm2). author(subm2,"teit"). author(subm2,"gian").} \\
\text{author(subm2,"rsch"). author(subm2,"apol").}
\end{align*}
\]

The program committee:

\[
\begin{align*}
\text{pc("vlif"). pc("mgel"). pc("dfen"). pc("fley"). pc("smil").} \\
\text{pc("mkif"). pc("ptra"). pc("ggot"). pc("ihor").}
\end{align*}
\]

All PC members are in the “Club100” with more than 100 papers:
Consider all senior researchers as candidate reviewers adding the club100 information
to the OWL knowledge base:

\[
cand(X,P) :- \text{paper}(P), \text{DL["club100" += pc;"senior"][X].}
\]
Facts:

\[
\text{paper(subm1). author(subm1,"jdbr"). author(subm1,"htom").} \\
\text{paper(subm2). author(subm2,"teit"). author(subm2,"gian").} \\
\text{author(subm2,"rsch"). author(subm2,"apol").}
\]

The program committee:

\[
\text{pc("vlif"). pc("mgel"). pc("dfen"). pc("fley"). pc("smil").} \\
\text{pc("mkif"). pc("ptra"). pc("ggot"). pc("ihor").}
\]

All PC members are in the “Club100” with more than 100 papers:
Consider all senior researchers as candidate reviewers adding the club100 information to the OWL knowledge base:

\[
cand(X,P) :- \text{paper(P), DL["club100" += pc;"senior"}(X).
\]
Guess a reviewer assignment:

assign(X,P) :- not -assign(X,P), cand(X,P).
\[-assign(X,P) :- not assign(X,P), cand(X,P).\]

Check that each paper is assigned to at most one person:

\[
\text{:- assign}(X,P), \text{assign}(X1,P), X1 \neq X.
\]

A reviewer can’t review a paper by him/herself:

\[
\text{:- assign}(A,P), \text{author}(P,A).
\]

Check whether all papers are correctly assigned (by projection)

\[
a(P) :- \text{assign}(X,P).
\]
\[
\text{error}(P) :- \text{paper}(P), \text{not a}(P).
\]
\[
\sim \text{error}(P).
\]

Note: error(P) detects unassignable papers rather than a simple constraint.
Review Assignment /3

Guess a reviewer assignment:

\[
\text{assign}(X,P) :- \neg \text{-assign}(X,P), \text{cand}(X,P).
\]
\[
\neg\text{-assign}(X,P) :- \neg \text{assign}(X,P), \text{cand}(X,P).
\]

Check that each paper is assigned to at most one person:

\[
:- \text{assign}(X,P), \text{assign}(X1,P), X1 \neq X.
\]

A reviewer can’t review a paper by him/herself:

\[
:- \text{assign}(A,P), \text{author}(P,A).
\]

Check whether all papers are correctly assigned (by projection)

\[
\text{a}(P) :- \text{assign}(X,P).
\]
\[
\text{error}(P) :- \text{paper}(P), \neg \text{a}(P).
\]
\[
\neg\text{error}(P).
\]

Note: \text{error}(P) detects unassignable papers rather than a simple constraint.
Review Assignment /3

Guess a reviewer assignment:

assign(X,P) :- not ~assign(X,P), cand(X,P).
~assign(X,P) :- not assign(X,P), cand(X,P).

Check that each paper is assigned to at most one person:

:- assign(X,P), assign(X1,P), X1 != X.

A reviewer can’t review a paper by him/herself:

:- assign(A,P), author(P,A).

Check whether all papers are correctly assigned (by projection)

a(P) :- assign(X,P).
error(P) :- paper(P), not a(P).
~error(P).

Note: error(P) detects unassignable papers rather than a simple constraint.
Guess a reviewer assignment:

assign(X,P) :- not -assign(X,P), cand(X,P).
-assign(X,P) :- not assign(X,P), cand(X,P).

Check that each paper is assigned to at most one person:

:- assign(X,P), assign(X1,P), X1 != X.

A reviewer can’t review a paper by him/herself:

:- assign(A,P), author(P,A).

Check whether all papers are correctly assigned (by projection)

a(P) :- assign(X,P).
error(P) :- paper(P), not a(P).
~ error(P).

Note: error(P) detects unassignable papers rather than a simple constraint.
Review Assignment /3

Guess a reviewer assignment:

\[
\text{assign}(X, P) :- \neg \text{assign}(X, P), \text{cand}(X, P).
\]
\[
\neg\text{assign}(X, P) :- \neg \text{assign}(X, P), \text{cand}(X, P).
\]

Check that each paper is assigned to at most one person:

\[
:- \text{assign}(X, P), \text{assign}(X_1, P), X_1 \neq X.
\]

A reviewer can’t review a paper by him/herself:

\[
:- \text{assign}(A, P), \text{author}(P, A).
\]

Check whether all papers are correctly assigned (by projection)

\[
\text{a}(P) :- \text{assign}(X, P).
\]
\[
\text{error}(P) :- \text{paper}(P), \neg \text{a}(P).
\]
\[
\neg\text{error}(P).
\]

Note: \text{error}(P) detects unassignable papers rather than a simple constraint.
Task

*Try out the complete reviewer example!*

Run *reviewer.dlp*!