Answer Set Programming for the Semantic Web

Tutorial

Technische Universität Wien
Vienna University of Technology
Università della Calabria
Campus di Arcavacata
Universidad Rey Juan Carlos

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.

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.

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.

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.

- 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
  $\Rightarrow$ 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*

$\Rightarrow$ 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/

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
  $\Rightarrow$ 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*

$\Rightarrow$ 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.

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

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

  add all assertions $DryWine(c)$ 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

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

  add all assertions $DryWine(c)$ 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

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

  add all assertions $DryWine(c)$ to the ABox (extensional part) of L, such that $my\_dry(c)$ holds.

dl-Atoms /2

A dl-*atom* has the form

$$DL[S_1\, op_1 p_1, \ldots, S_m\, op_m\, p_m; Q](\mathbf{t}), \qquad m \geq 0,$$

where

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

Intuitively:

$op_i = \uplus$ increases S_i by p_i.
$op_i = \mathbin{\cup\!\!\!\!\!-}$ increases $\neg S_i$ by p_i.

dl-Atoms /2

A dl-*atom* has the form

$$DL[S_1\, op_1 p_1, \ldots, S_m\, op_m\, p_m; Q](\mathbf{t})\,, \qquad m \geq 0,$$

where

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

Intuitively:

$op_i = \uplus$ increases S_i by p_i.
$op_i = \cup\!\!\!-$ increases $\neg S_i$ by p_i.

dl-Atoms /2

A dl-*atom* has the form

$$DL[S_1\, op_1\, p_1, \ldots, S_m\, op_m\; p_m; Q](\mathbf{t}), \qquad m \geq 0,$$

where

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

Intuitively:

$op_i = \uplus$ increases S_i by p_i.
$op_i = \uminus$ increases $\neg S_i$ by p_i.

dl-Atoms /2

A dl-*atom* has the form

$$DL[S_1\, op_1 p_1, \ldots, S_m\, op_m\; p_m; Q](\mathbf{t})\,, \qquad m \geq 0,$$

where

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

Intuitively:

$op_i = \uplus$ increases S_i by p_i.
$op_i = \cup\!\!\!\!-$ increases $\neg S_i$ by p_i.

dl-Atoms /2

A dl-*atom* has the form

$$DL[S_1\, op_1 p_1, \ldots, S_m\, op_m\; p_m; Q](\mathbf{t})\,, \qquad m \geq 0,$$

where

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

Intuitively:

$op_i = \uplus$ increases S_i by p_i.
$op_i = \cup\!\!\!\!-$ increases $\neg S_i$ by p_i.

DL Queries

A *DL query* $Q(\mathbf{t})$ 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.

dl-Programs

A dl-*rule* r is of form

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

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* $(\mathrm{dl}\text{-})$ *program* $KB = (L, P)$ consists of

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

dl-Programs

A dl-*rule* r is of form

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

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* $(\mathrm{dl}\text{-})$ *program* $KB = (L, P)$ consists of

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

Social Dinner IX

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

Social Dinner IX

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]:

```
white("SelaksIceWine"). not_dry("SelaksIceWine").
```

- In our program, we may pass this information to the ontology by adding in the dl-atoms the modification

$$WhiteWine \uplus white, DryWine \cupdot not\_dry.$$

E.g., `DL[Wine](X)` is changed to

```
DL[WhiteWine += white, DryWine -= not_dry; Wine](X).
```

[1] See wineCover09c.dlp

Semantics of $KB = (L, P)$

- HB_P^Φ: Set of all ground (classical) literals with predicate symbol in P and constants from finite relational alphabet Φ.
- 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 ℓ, iff $\ell \in I$;
  - $I \models_L DL[S_1\, op_1\, p_1 \ldots, S_m\, op_m\, p_m; Q](\mathbf{c})$ if and only if

$$L \cup A_1(I) \cup \cdots \cup A_m(I) \models Q(\mathbf{c}),$$

    where
    - $A_i(I) = \{S_i(\mathbf{e}) \mid p_i(\mathbf{e}) \in I\}$, for $op_i = \uplus$;
    - $A_i(I) = \{\neg S_i(\mathbf{e}) \mid p_i(\mathbf{e}) \in I\}$, 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^Φ: Set of all ground (classical) literals with predicate symbol in P and constants from finite relational alphabet Φ.
- 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 ℓ, iff $\ell \in I$;
  - $I \models_L DL[S_1\, op_1\, p_1 \ldots, S_m\, op_m\, p_m; Q](\mathbf{c})$ if and only if

$$L \cup A_1(I) \cup \cdots \cup A_m(I) \models Q(\mathbf{c}),$$

    where
    - $A_i(I) = \{S_i(\mathbf{e}) \mid p_i(\mathbf{e}) \in I\}$, for $op_i = \uplus$;
    - $A_i(I) = \{\neg S_i(\mathbf{e}) \mid p_i(\mathbf{e}) \in I\}$, 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^Φ: Set of all ground (classical) literals with predicate symbol in P and constants from finite relational alphabet Φ.
- 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 ℓ, iff $\ell \in I$;
  - $I \models_L DL[S_1\, op_1\, p_1 \ldots, S_m\, op_m\, p_m; Q](\mathbf{c})$ if and only if

    $$L \cup A_1(I) \cup \cdots \cup A_m(I) \models Q(\mathbf{c}),$$

    where
    - $A_i(I) = \{S_i(\mathbf{e}) \mid p_i(\mathbf{e}) \in I\}$, for $op_i = \uplus$;
    - $A_i(I) = \{\neg S_i(\mathbf{e}) \mid p_i(\mathbf{e}) \in I\}$, 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^Φ: Set of all ground (classical) literals with predicate symbol in P and constants from finite relational alphabet Φ.
- 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 ℓ, iff $\ell \in I$;
  - $I \models_L DL[S_1\, op_1\, p_1 \ldots, S_m\, op_m\, p_m; Q](\mathbf{c})$ if and only if

$$L \cup A_1(I) \cup \cdots \cup A_m(I) \models Q(\mathbf{c}),$$

  where
    - $A_i(I) = \{S_i(\mathbf{e}) \mid p_i(\mathbf{e}) \in I\}$, for $op_i = \uplus$;
    - $A_i(I) = \{\neg S_i(\mathbf{e}) \mid p_i(\mathbf{e}) \in I\}$, for $op_i = \cup\!\!\!-$.
- The models of $KB = (L, P)$ are the joint models of all rules in P (defined as usual)

Examples

- Suppose $L \models \mathit{Wine}(\text{“}\mathit{TaylorPort}\text{”})$, and I contains $\mathit{wineBottle}(\text{“}\mathit{TaylorPort}\text{”})$

  Then $I \models_L DL[\text{“}\mathit{Wine}\text{”}](\text{“}\mathit{TaylorPort}\text{”})$ and

  $$I \models_L \mathit{wineBottle}(\text{“}\mathit{TaylorPort}\text{”}) \text{ :- } DL[\text{“}\mathit{Wine}\text{”}](\text{“}\mathit{TaylorPort}\text{”})$$

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

  Then $I \models_L$
  $DL[\text{“}\mathit{WhiteWine}\text{”} \uplus \mathit{white}, \text{“}\mathit{DryWine}\text{”} \cupdot \mathit{not\_dry}; \text{“}\mathit{Wine}\text{”}](\text{“}\mathit{siw}\text{”})$

Examples

- Suppose $L \models Wine(\text{"}TaylorPort\text{"})$, and I contains $wineBottle(\text{"}TaylorPort\text{"})$

  Then $I \models_L DL[\text{"}Wine\text{"}](\text{"}TaylorPort\text{"})$ and

  $$I \models_L wineBottle(\text{"}TaylorPort\text{"}) \text{ :- } DL[\text{"}Wine\text{"}](\text{"}TaylorPort\text{"})$$

- Suppose $I = \{white(\text{"}siw\text{"}),\ not\_dry(\text{"}siw\text{"})\}$.

  Then $I \models_L$ $DL[\text{"}WhiteWine\text{"} \uplus white, \text{"}DryWine\text{"} \cup\!\!\!\!-\, not\_dry; \text{"}Wine\text{"}](\text{"}siw\text{"})$

Examples /2

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

$$I \models_L compliant(joe, \text{“}Milk\text{”}) \text{ :- } DL[\text{“}Wine\text{”}](\text{“}Milk\text{”})$$

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

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

- Inconsistency of L is revealed with unsatisfiable DL queries:

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

Shorthand: $DL[\bot]$

- Consistency can be checked by

$$consistent \text{ :- } not\ DL[\text{“}Wine\text{”} \sqsubseteq \neg\text{“}Wine\text{”}]$$

Examples /2

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

$$I \models_L compliant(joe, \text{“}Milk\text{”}) \text{ :- } DL[\text{“}Wine\text{”}](\text{“}Milk\text{”})$$

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

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

- Inconsistency of L is revealed with unsatisfiable DL queries:

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

Shorthand: $DL[\bot]$

- Consistency can be checked by

$$consistent \text{ :- } not\ DL[\text{“}Wine\text{”} \sqsubseteq \neg\text{“}Wine\text{”}]$$

Examples /2

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

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

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

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

- Inconsistency of L is revealed with unsatisfiable DL queries:

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

Shorthand: $DL[\bot]$

- Consistency can be checked by

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

Examples /2

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

$$I \models_L compliant(joe, \text{“}Milk\text{”}) \text{ :- } DL[\text{“}Wine\text{”}](\text{“}Milk\text{”})$$

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

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

- Inconsistency of L is revealed with unsatisfiable DL queries:

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

  Shorthand: $DL[\bot]$

- Consistency can be checked by

$$consistent \text{ :- } not \; DL[\text{“}Wine\text{”} \sqsubseteq \neg\text{“}Wine\text{”}]$$

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 $I = lm(KB^I)$.

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 "$\neg$" and constraints always have an answer set
- *Uniqueness:* Layered use of "not" (stratified dl-program) $\Rightarrow$ 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 "$\neg$" and constraints always have an answer set
- *Uniqueness:* Layered use of "not" (stratified dl-program) $\Rightarrow$ 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 "$\neg$" and constraints always have an answer set
- *Uniqueness:* Layered use of "not" (stratified dl-program) $\Rightarrow$ 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 "$\neg$" and constraints always have an answer set
- *Uniqueness:* Layered use of "not" (stratified dl-program) $\Rightarrow$ 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 "$\neg$" and constraints always have an answer set
- *Uniqueness:* Layered use of "not" (stratified dl-program) $\Rightarrow$ 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

- 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

- 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

- 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 $KB \not\models p(c)$

- Express CWA for concepts $C_1, \ldots, C_k$ wrt. individuals in L:

$$\begin{array}{rcl} \neg c_1(X) & \leftarrow & not\ DL[C_1](X) \\ & \ldots & \\ \neg c_k(X) & \leftarrow & not\ DL[C_k](X) \end{array}$$

- CWA for roles R: easy extension

Query Answering under CWA

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

Query: $\mathit{WhiteWine}(\text{“}\mathit{VeuveCliquot}\text{”})$ (Y/N)?

Query Answering under CWA

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

Query: $WhiteWine(\text{“}VeuveCliquot\text{”})$ (Y/N)?

Add CWA-literals to L:

$$
\begin{array}{rcl}
\overline{sp}(X) & \leftarrow & not\ \ DL[SparklingWine](X) \\
\overline{ww}(X) & \leftarrow & not\ \ DL[WhiteWine](X) \\
ww(X) & \leftarrow & DL[SparklingWine \uplus \overline{sp}, \\
 & & \quad WhiteWine \uplus \overline{ww};\ WhiteWine](X)
\end{array}
$$

Ask whether $KB \models ww(\text{“}VeuveCliquot\text{”})$ or
$KB \models \overline{ww}(\text{“}VeuveCliquot\text{”})$

Extended CWA

- CWA can be inconsistent (disjunctive knowledge)
- **Example:**
  Knowledge base

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

  - CWA for $Painter, Singer$ adds

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

  - This implies $\neg Artist(\text{“}Jody\text{”})$

Extended CWA

- CWA can be inconsistent (disjunctive knowledge)
- **Example:**
  Knowledge base

$$L = \{ \, Artist(\text{``}Jody\text{''}), Artist \equiv Painter \sqcup Singer \, \}$$

  - CWA for $Painter, Singer$ adds

$$\neg Painter(\text{``}Jody\text{''}), \neg Singer(\text{``}Jody\text{''}).$$

  - This implies $\neg Artist(\text{``}Jody\text{''})$

Minimal Models

- ECWA singles out "minimal" models of L wrt $Painter$ and $Singer$ (UNA in L on ABox):

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

Answer sets:

$$\begin{aligned}
M_1 &= \{p(\text{“}Jody\text{”}), \overline{s}(\text{“}Jody\text{”})\},\\
M_2 &= \{s(\text{“}Jody\text{”}), \overline{p}(\text{“}Jody\text{”})\}
\end{aligned}$$

- Extendible to keep concepts "fixed"
$\rightsquigarrow$ ECWA$(\phi; P; Q; Z)$

Minimal Models

- ECWA singles out "minimal" models of L wrt $Painter$ and $Singer$ (UNA in L on ABox):

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

  Answer sets:

$$\begin{aligned} M_1 &= \{p(\text{``}Jody\text{''}), \overline{s}(\text{``}Jody\text{''})\}, \\ M_2 &= \{s(\text{``}Jody\text{''}), \overline{p}(\text{``}Jody\text{''})\} \end{aligned}$$

- Extendible to keep concepts "fixed"
  $\rightsquigarrow$ ECWA$(\phi; P; Q; Z)$

Minimal Models

- ECWA singles out "minimal" models of L wrt $Painter$ and $Singer$ (UNA in L on ABox):

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

Answer sets:

$$
\begin{aligned}
M_1 &= \{p(\text{"}Jody\text{"}), \overline{s}(\text{"}Jody\text{"})\}, \\
M_2 &= \{s(\text{"}Jody\text{"}), \overline{p}(\text{"}Jody\text{"})\}
\end{aligned}
$$

- Extendible to keep concepts "fixed"
$\leadsto$ ECWA$(\phi; P; Q; Z)$

Default Reasoning

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

Example: wine ontology

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

Use default rule: Sparkling wines are white by default

$$\begin{array}{lrcl}
r1: & \mathit{white}(W) & \leftarrow & DL[\mathit{SparklingWine}](W), \mathit{not}\ \neg\mathit{white}(W) \\
r2: & \neg\mathit{white}(W) & \leftarrow & DL[\mathit{WhiteWine} \uplus \mathit{white}; \neg\mathit{WhiteWine}](W) \\
r3: & f & \leftarrow & \mathit{not}\ \ f, DL[\bot] \quad \text{/* kill model if } L \text{ is inconsistent */}
\end{array}$$

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

Default Reasoning

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

Example: wine ontology

$$L = \{\ \mathit{SparklingWine}(\text{``}\mathit{VeuveCliquot}\text{''}), \\ (\text{``}\mathit{SparklingWine}\text{''} \sqcap \neg \text{``}\mathit{WhiteWine}\text{''})(\text{``}\mathit{Lambrusco}\text{''})\ \},$$

Use default rule: Sparkling wines are white by default

$$\begin{array}{lrcl} r1: & \mathit{white}(W) & \leftarrow & \mathit{DL}[\mathit{SparklingWine}](W), \mathit{not}\ \neg \mathit{white}(W) \\ r2: & \neg \mathit{white}(W) & \leftarrow & \mathit{DL}[\mathit{WhiteWine} \uplus \mathit{white}; \neg \mathit{WhiteWine}](W) \\ r3: & f & \leftarrow & \mathit{not}\ \ f,\ \mathit{DL}[\bot] \quad /\text{* kill model if } L \text{ is inconsistent */} \end{array}$$

- In answer set semantics, $r2$ effects maximal application of $r1$.
- Answer Set: $M = \{\mathit{white}(\text{``}\mathit{VeuveCliquot}\text{''}),\ \neg \mathit{white}(\text{``}\mathit{Lambrusco}\text{''})\}$

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 $\cap\!\!\!-$

$$DL[WhiteWine \cap\!\!\!- my\_WhiteWine](X)$$

  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 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 $\cap\!\!\!\!-$

  $$DL[WhiteWine \cap\!\!\!\!- my\_WhiteWine](X)$$

  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 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 $\cap\!\!\!\!-$

$$DL[WhiteWine \cap\!\!\!\!- my\_WhiteWine](X)$$

  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 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 $\cap\!\!\!\!-$

  $$DL[WhiteWine \cap\!\!\!\!- my\_WhiteWine](X)$$

  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.

Computational Complexity

Deciding strong answer set existence for dl-programs (completeness results)

$KB = (L, P)$	L in $\mathcal{SHIF}(\mathbf{D})$	L in $\mathcal{SHOIN}(\mathbf{D})$
positive	EXP	NEXP
stratified	EXP	$\mathrm{P}^{\mathrm{NEXP}}$
general	NEXP	$\mathrm{NP}^{\mathrm{NEXP}}$

Recall: Satisfiability problem in

- $\mathcal{SHIF}(\mathbf{D})$ / $\mathcal{SHOIN}(\mathbf{D})$ 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
- $\mathrm{NP}^{\mathrm{NEXP}} = \mathrm{P}^{\mathrm{NEXP}}$ is less powerful than disjunctive ASP ($\equiv \mathrm{NEXP}^{\mathrm{NP}}$)
- 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 $\leftarrow$, $\uplus$, and $\cup\!\!\!-$, 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:

```
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:

```
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 /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.

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.

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.

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.

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.

Task

Try out the complete reviewer example!

Run reviewer.dlp !