

Supporting Open and Closed World Reasoning in the Semantic Web

C. V. Damásio, A. Analyti,
G. Antoniou, G. Wagner

Overview

- Motivation
- Open and Closed World Reasoning
- Building blocks
- Knowledge in the Semantic Web
- Composing modules together
- Transformational Semantics
- Conclusions

Motivation

- Merging knowledge in the Semantic Web is one **fundamental** unsolved problem
- The need of combining closed and open world reasoning is **desirable**
- The adopted mechanisms should be **modular**
- The solution should be **independent** of the semantics adopted
- Use of nonmonotonic reasoning in the Semantic Web should be carefully **controlled**

Approach

- Open and closed world assumptions can be **already combined** in extended logic programming!
- It is required **two forms** of negation:
 - strong or explicit
 - weak, default or *as failure*
- The two forms of negation are available in
 - Well-founded semantics with explicit negation (WFSX)
 - Answer Set Semantics (AS)
- The proposed solution uses the **same** program transformation for both semantics
- The user should have an easy **syntactic mechanisms** to specify the use of nonmonotonic reasoning constructs

Open and Closed World reasoning

- Open World Reasoning
 - Founded on First Order Logic
 - Adopted in Description Logics, OWL and SWRL
 - Appropriate for the Semantic Web
 - Sometimes too conservative
- Closed World Reasoning
 - Founded on Nonmonotonic Logics
 - Adopted in Logic Programming and WRL
 - Appropriate for (Deductive) Databases
 - Sometimes too brave

Example

- Consider the following list of facts

% All current EU countries

CountryEU(Austria) ... CountryEU(UK)

% Some non EU countries (not all...)

¬ CountryEU(China)

¬ CountryEU(Djibuti)

A little geography...

- Is Austria a EU country ?
 - **YES**, because it appears the fact $\text{CountryEU}(\text{Austria})$ in the knowledge base
- Is China a EU country ?
 - **NO**, because it is expressed that $\neg \text{CountryEU}(\text{China})$
- Is Montenegro a EU country ?
 - **NO**, because it is not listed there and the list is complete (CLOSED WORLD REASONING)
 - **DON'T KNOW**, since it is not listed then it might be or not (OPEN WORLD REASONING)

The help of extended LP

- Closed world reasoning:

$$\neg \text{CountryEU}(?C) \leftarrow \sim \text{CountryEU}(?C)$$

- Open world reasoning:

$$\neg \text{CountryEU}(?C) \leftarrow \sim \text{CountryEU}(?C)$$

$$\text{CountryEU}(?C) \leftarrow \sim \neg \text{CountryEU}(?C)$$

A syntactic detour

- Rule bases are sets of rules of the form
 - $L_0 \leftarrow L_1, \dots, L_m, \sim L_{m+1}, \dots, \sim L_n$
- Each L_i ($0 \leq i \leq n$) is an objective literal, i.e.
 - An atom $A(t)$, or
 - The strong negation of an atom $\neg A(t)$
- The symbol \sim represents nonmonotonic weak negation, and cannot occur in the head
- The symbol \neg represents monotonic strong negation, and can occur in the head and in the body of rules
- The discussion is restricted to the DATALOG case, i.e. no function symbols in the language

Putting weak negation on the leash

- The following predicate types are proposed
 - Definite or objective predicates
 - Open predicates
 - Closed predicates
 - Normal or unrestricted predicates
- Definite, open and closed predicates are limited to be defined by rules **without** weak negation
- Normal predicates can use the full language

Definite Predicates

- **Similar to** Definite Logic Programming, but allowing for explicit negation in the head and body of rules
- There can exist information gaps: predicates are partial
- Reasoning is purely monotonic
- Reasoning is polynomial on the size of the ground rule base and can be readily implemented in Prolog

Open Predicates

- Rules are like in the previous case, but additionally it is added the following pair of rules for each open predicate A with arity n
 - $\neg A(?x_1, \dots, ?x_n) \leftarrow \sim A(?x_1, \dots, ?x_n)$
 - $A(?x_1, \dots, ?x_n) \leftarrow \sim \neg A(?x_1, \dots, ?x_n)$
- Reasoning is monotonic
- Reasoning is polynomial for WFSX and co-NP complete for AS
- Can be implemented with XSB or any answer set programming system like DLV, Smodels, etc.

Closed Predicates

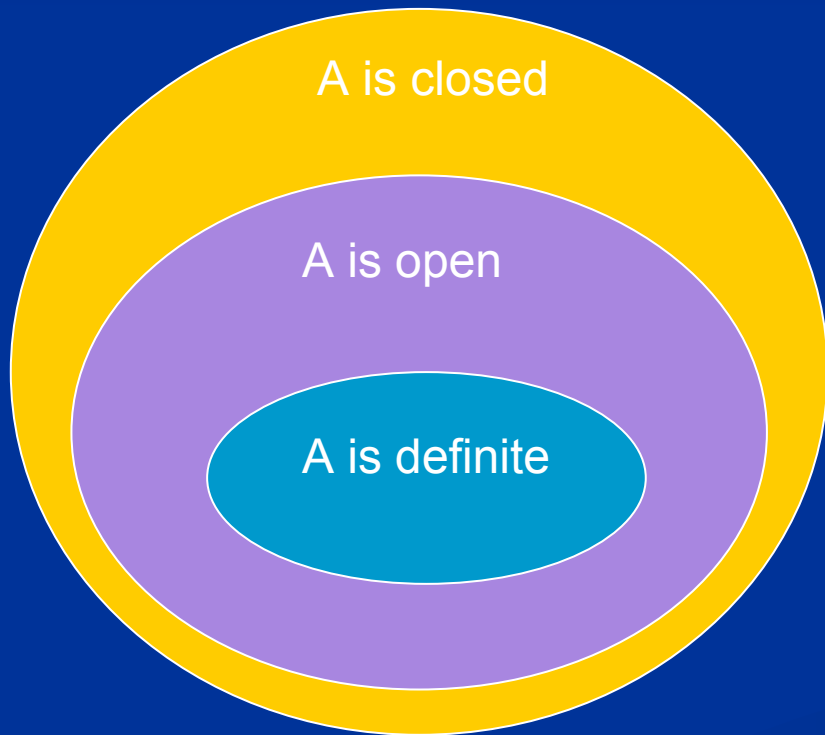
- Rules are like in the previous case, but it is added **only one** of the following pair of rules for closed predicate A with arity n
 - $\neg A(?x_1, \dots, ?x_n) \leftarrow \sim A(?x_1, \dots, ?x_n)$
 - $A(?x_1, \dots, ?x_n) \leftarrow \sim \neg A(?x_1, \dots, ?x_n)$
- Reasoning is nonmonotonic
- Conclusions obtained by objective predicates are also obtained by closed ones (common safe knowledge)

Normal Predicates

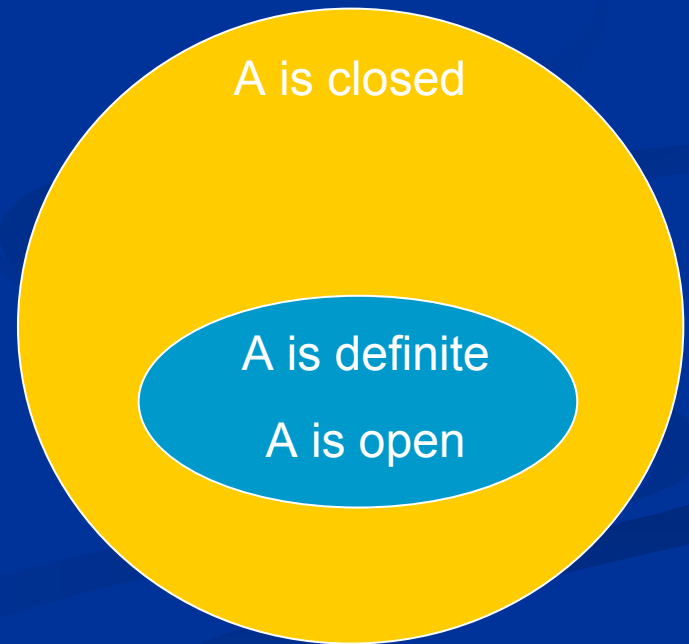
- Full syntax of extended logic programming
- Nonmonotonic
- No guarantees...
- Sometimes it is required

Entailment of Objective Literals

- Predicates are all definite or open, except varying A



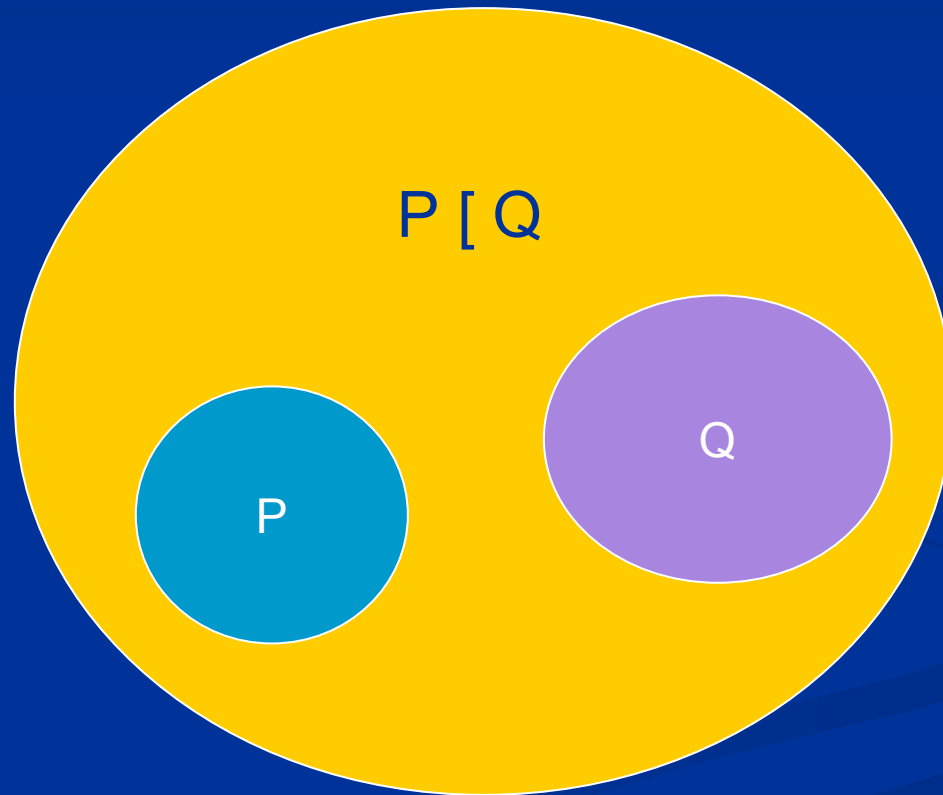
Answer Set Semantics



Well-founded Semantics with Explicit Negation

Monotonicity of Reasoning

- *All* predicates in rule bases P and Q are either definite or open



Particularities of the Semantic Web

- Rule bases cannot be seen isolated
- Modularity, encapsulation, information hiding and access control mechanisms are required
- Have to deal with four levels of context
 - Semantic Web context
 - Application context
 - Rule base context
 - Predicate context
- IRIs should be used in names of rule bases and predicates

Requirements

- Applications loading/asserting knowledge need mechanisms to express that
 - Nonmonotonic reasoning is allowed or inhibited
 - Force to use only safe knowledge
- Producers of knowledge need mechanisms to
 - Declare that a predicate cannot be redefined
 - Declare hidden predicates not visible in the Semantic Web
 - Use all available knowledge in the application context, or get it explicitly from particular rule bases
- Monotonic reasoning is the default, not the exception

DEFINES and USES declaration

- The **DEFINES** declaration states
 - The predicates defined in a rule base and their type
 - The predicates exported and their scope
- The **USES** declaration states
 - The predicates imported, and from where
 - Reasoning mode to be used

DEFINES

| | |
|----------------------|--|
| <i>[RuleBaseIRI]</i> | (Absolute IRI, default is the rule base where it occurs) |
| defines | |
| <i>[ScopeDecl]</i> | global local internal |
| <i>PredDeclList</i> | (objective open closed [¬] normal) <i>AbsIRI</i> [/N], ... |
| [visible to | |
| <i>RuleBaseList]</i> | (list of Absolute IRIs, if omitted, visible everywhere) |

USES declaration

| | |
|----------------------|---|
| <i>[RuleBaseIRI]</i> | (Absolute IRI, default is the rule base where it occurs) |
| uses | |
| <i>PredDeclList</i> | (objective open closed [\neg] normal) <i>AbsIRI</i> [/N], ... |
| [from | |
| <i>RuleBaseList]</i> | (list of Abs. IRIs, by default uses from any available rulebase) |

NOTE: The scope of an imported predicate is given by a corresponding **defines** declaration. If absent, the predicate is **global and open**; the defaults adopted!

Combining reasoning forms

| | | | | | |
|---------------------------|-----------|---------------------------|-----------|-----------|--------|
| uses (importer) | normal | objective | open | closed | normal |
| | closed | objective | open | closed | error |
| | open | objective | open | open | error |
| | objective | objective | objective | objective | error |
| | | objective | open | closed | normal |
| | | defines (exporter) | | | |

Defining and using the same predicate

uses

| | | | |
|----------|---------|-------|-----------------|
| global | allowed | error | allowed |
| local | error | error | allowed |
| internal | error | error | error |
| | global | local | <u>internal</u> |

defines

Example

■ `<http://www.eu.int>`

defines local closed `eu:CountryEU/1`.

`eu:CountryEU(Austria)`

```
d_<http://www.eu.int><http://www.eu.int#CountryEU>(Austria).
o_<http://www.eu.int><http://www.eu.int#CountryEU>(Austria).
c_<http://www.eu.int><http://www.eu.int#CountryEU>(Austria).
n_<http://www.eu.int><http://www.eu.int#CountryEU>(Austria).
```

```
d_<http://www.eu.int><http://www.eu.int#CountryEU>(X) :- (X).
-d_<http://www.eu.int#CountryEU>(X) :- ~ - o_<http://www.eu.int><http://www.eu.int#CountryEU>(X). >(X).
-o_<http://www.eu.int><http://www.eu.int#CountryEU>(X) :-
~ o_<http://www.eu.int><http://www.eu.int#CountryEU>(X). (X).
-o_<http://www.eu.int#CountryEU>(X) :- >(X).
-c_<http://www.eu.int><http://www.eu.int#CountryEU>(X) :-
~ c_<http://www.eu.int><http://www.eu.int#CountryEU>(X). (X).
-c_<http://www.eu.int#CountryEU>(X) :- >(X).
-n_<http://www.eu.int>n_CountryEU(X) :-
~ n_<http://www.eu.int><http://www.eu.int#CountryEU>(X). (X).
-n_<http://www.eu.int#CountryEU>(X) :- -n_<http://www.eu.int><http://www.eu.int#CountryEU>(X).
```

Example

■ <http://security.int>

defines global open sec#citizenOf/2.

sec:citizenOf(Arne, Austria).

sec:citizenOf(Boris, Bulgaria).

sec:citizenOf/2 <http://security.int> <http://security.int#citizenOf>(X,Y) :-

~ - o_ <http://security.int> <http://security.int#citizenOf>(X,Y).

sec:citizenOf/2 -o_ <http://security.int> <http://security.int#citizenOf>(X,Y) :-

~ o_ <http://security.int> <http://security.int#citizenOf>(X,Y).

c_ <http://security.int> <http://security.int#citizenOf>(X,Y) :-

~ - c_ <http://security.int> <http://security.int#citizenOf>(X,Y).

-c_ <http://security.int> <http://security.int#citizenOf>(X,Y) :-

~ c_ <http://security.int> <http://security.int#citizenOf>(X,Y).

n_ <http://security.int> n_citizenOf(X,Y) :-

~ - n_ <http://security.int> <http://security.int#citizenOf>(X,Y).

-n_ <http://security.int> n_citizenOf(X,Y) :-

~ n_ <http://security.int> <http://security.int#citizenOf>(X,Y).

Example

■ <http://gov.coun

defines local closed
defines internal obje
defines internal clos
defines internal
uses objective eu:C
defines internal obje
uses sec:citizenOf/2

```
d_<http://gov.country><http://security.int#citizenOf>(X,Y) :-  
    d_<http://security.int#citizenOf>(X,Y).  
-d_<http://gov.country><http://security.int#citizenOf>(X,Y) :-  
    - d_<http://security.int#citizenOf>(X,Y).  
  
o_<http://gov.country><http://security.int#citizenOf>(X,Y) :-  
    o_<http://security.int#citizenOf>(X,Y).  
-o_<http://gov.country><http://security.int#citizenOf>(X,Y) :-  
    - o_<http://security.int#citizenOf>(X,Y).  
  
c_<http://gov.country><http://security.int#citizenOf>(X,Y) :-  
    o_<http://security.int#citizenOf>(X,Y).  
-c_<http://gov.country><http://security.int#citizenOf>(X,Y) :-  
    - o_<http://security.int#citizenOf>(X,Y).  
  
n_<http://gov.country><http://security.int#citizenOf>(X,Y) :-  
    o_<http://security.int#citizenOf>(X,Y).  
-n_<http://gov.country><http://security.int#citizenOf>(X,Y) :-  
    - o_<http://security.int#citizenOf>(X,Y).
```

Example

gov:Enter(?p) ← eu:CountryEU(?c), sec:citizenOf(?p,?c).

gov:Enter(?p) ← ¬ eu:CountryEU(?c), sec:citizenOf(?p,?c).

gov:Enter(?p)

⌈ gov:Require

⌈ gov:Require

gov:RequiresV

gov:HasVisa(

gov:HasVisa(

⌈ eu:CountryE

⌈ eu:Country

d_<http://gov.country><http://gov.country#Enter>(P) :-

d_<http://gov.country><http://www.eu.int#CountryEU>(C),

d_<http://gov.country><http://security.int#citizenOf>(P,C).

o_<http://gov.country><http://gov.country#Enter>(P) :-

o_<http://gov.country><http://www.eu.int#CountryEU>(C),

o_<http://gov.country><http://security.int#citizenOf>(P,C).

c_<http://gov.country><http://gov.country#Enter>(P) :-

c_<http://gov.country><http://www.eu.int#CountryEU>(C),

c_<http://gov.country><http://security.int#citizenOf>(P,C).

n_<http://gov.country><http://gov.country#Enter>(P) :-

n_<http://gov.country><http://www.eu.int#CountryEU>(C),

n_<http://gov.country><http://security.int#citizenOf>(P,C).

Summary of the program transformation

- Each rule is translated into four different rules, one for each reasoning mode
 - Definite (prefix d)
 - Open (prefix o)
 - Closed (prefix c)
 - Normal (prefix n)
- The predicate name is obtained by composition of the IRI of the Rule base plus prefix and the IRI of the predicate name
- The rules for open and closed predicates are as well introduced.
- Global and local predicates introduce a rule with prefix plus the IRI of the predicate, in order to make it visible everywhere
- USES declarations are introduced by respecting combination of reasoning forms in the table.

Problems to be addressed

- Implicit Domain Closure Assumption
- Unique Names Assumption
- Expressing the domain of predicates in order to avoid floundering, using for instance `rdf:domain` and `rdf:range`
- Optimisation of the program transformation: too many repeated rules
- Handling disjunction and paraconsistency

Conclusions

- Modular approach to mixing open and closed world reasoning
- Users have mechanisms to control the use of nonmonotonic reasoning in the Semantic Web
- Defines the notion of scope of predicates
- Captures the intuitions of knowledge merging in the Semantic Web
- Solution based on widely accepted semantics
- Polynomial program transformation for WFSX and AS