Aalto University
School of Science

SeCo

Logic and Inference Rules on the Semantic Web

Eero Hyvönen
Aalto University, Semantic Computing Research Group (SeCo) *http://seco.cs.aalto.fi*
University of Helsinki, HELDIG *http://heldig.fi*

eero.hyvonen@aalto.fi

Learning Objectives

- Understand the role of logic on the Semantic Web
- Learn a variety of approaches to use logic rules

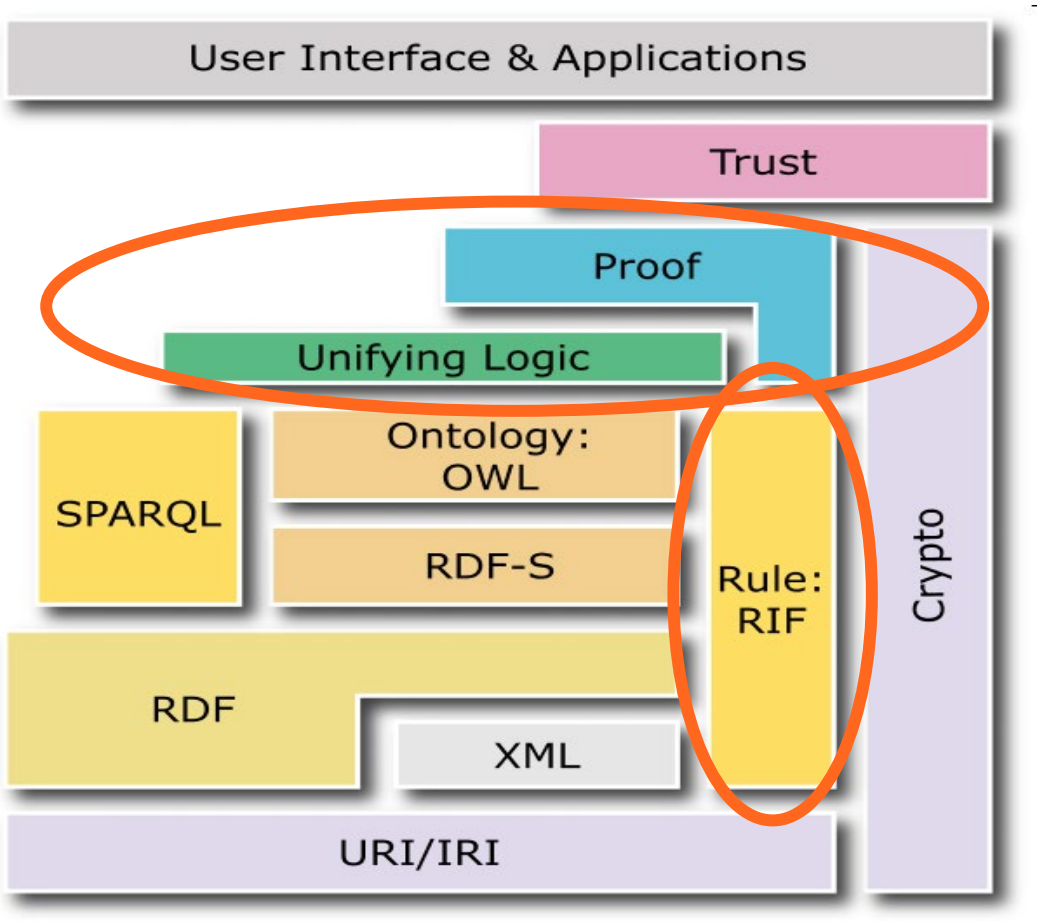
Contents

- Role of Logic Rules in the Semantic Web Stack
- Logic Rule Approaches in Use
- Interplay of Ontologies and Logical Rules
- Non-monotonic Rules

Logic Rules in the Semantic Web Stack

Semantic Web Technology Stack

”Layer Cake Model”



The Importance of Logic

- High-level language for expressing knowledge
- High expressive power
- Well-understood formal semantics
- Proof systems can automatically derive statements syntactically from a set of premises
- Sound & complete proof systems exist
  - *Subsets of First Order Predicate Logic*
  - *Not necessarily available for more expressive logics*
- Logic can provide **explanations** for answers
  - *Trace the proof that leads to a logical consequence*

Logic Rules: Many Approaches in Use

SPARQL-based rules

Logic programming using RDF data

- Using Horn Logic

Rule Interchange Format (RIF)

Semantic Web Rule Language SWRL

Ontologies and rules

Non-monotonic logics

SPARQL-based Rules

Idea

Use SPARQL CONSTRUCT for generating new triples

```
CONSTRUCT {
    ?this ex:grandParent ?grandParent .
}
WHERE {
    ?parent ex:child ?this .
    ?grandParent ex:child ?parent .
}
```

https://spinrdf.org/

SPIN

SPARQL Inferencing Notation

Overview

SPIN is a W3C Member Submission that has become the de-facto industry standard to represent SPARQL rules and constraints on Semantic Web models. SPIN also provides meta-modeling capabilities that allow users to define their own SPARQL functions and query templates. Finally, SPIN includes a ready to use library of common functions.

SPIN in Five Slides

For more information see SPIN Architecture and the SPIN Overview W3C Page.

Update (July 2017): Before you explore SPIN further, you may want to read **From SPIN to SHACL**.

What You Can Do with SPIN

SPIN is a way to represent a wide range of business rules.

You will not need to learn another proprietary rules language to do so. With SPIN, rules are expressed in SPARQL. In fact, SPIN is also referred to as *SPARQL Rules*. SPARQL is a well-established W3C standard implemented by many industrial-strength RDF APIs and all databases. This means that rules can run directly on RDF data without a need for materialization. SPIN provides a framework that helps users to leverage the fast performance and rich expressivity of SPARQL for various application purposes.

SPIN can be used to:

- **Calculate the value of a property based on other properties** - for example, area of a geometric figure as a product of its height and width, age of a person as a difference between today's date and person's birthday, a display name as a concatenation of the first and last names
- **Isolate a set of rules to be executed under certain conditions** - for example, to support incremental reasoning, to initialize certain values when a resource is first created, or to drive interactive applications

These rules are implemented USING SPARQL CONSTRUCT or SPARQL UPDATE requests (INSERT and DELETE). SPIN Templates also make it possible to define such rules in higher-level domain specific languages so that rule designers do not need to work with SPARQL directly.

Another common need in applications is to check validity of the data. For example, you may want to require that a field is entered and/or that the string entered follows your format requirements.

SPIN offers a way to do constraint checking with closed world semantics and automatically raise inconsistency flags when currently available information does not fit the specified integrity constraints. Constraints are specified using SPARQL ASK or CONSTRUCT queries, or corresponding SPIN Templates.

SPIN combines concepts from object oriented languages, query languages, and rule-based systems to describe object behavior on the web of data. One of the key ideas of SPIN is to link class definitions with SPARQL queries to capture rules and constraints that formalize the expected behavior of those classes. To do so, SPIN defines a light-weight collection of RDF properties.

Finally, SPIN also supports the definition of new SPARQL functions with a transparent and web-friendly framework.

Core Specifications

The SPIN SPARQL Syntax
The SPIN Modeling Vocabulary
The SPIN Standard Modules Library

Example of a Rule Using CONSTRUCT

New triples visible for the next rule (not inserted into data, but added into a special “inferences” graph)

```
ex:Person
  a       rdfs:Class ;
  rdfs:label "Person"^^xsd:string ;
  rdfs:subClassOf owl:Thing ;
  spin:rule
          [ a       sp:Construct ;
            sp:templates ([ sp:object sp:_grandParent ;
                            sp:predicate ex:grandParent ;
                            sp:subject spin:_this
                          ]) ;
            sp:where ([ sp:object spin:_this ;
                        sp:predicate ex:child ;
                        sp:subject sp:_parent
                      ] [ sp:object sp:_parent ;
                        sp:predicate ex:child ;
                        sp:subject sp:_grandParent
                      ])
          ] .
```

In textual SPARQL syntax, the above query would read as:

```
CONSTRUCT {
    ?this ex:grandParent ?grandParent .
}
WHERE {
    ?parent ex:child ?this .
    ?grandParent ex:child ?parent .
}
```

SPIN – SPARQL Inferencing Notation

For example, the SPARQL query

```
# must be at least 18 years old
ASK WHERE {
    ?this my:age ?age .
    FILTER (?age < 18) .
}
```

can be represented by a blank node in the SPIN RDF Syntax in Turtle as

```
[ a         sp:Ask ;
            rdfs:comment "must be at least 18 years old"^^xsd:string ;
            sp:where ([ sp:object sp:_age ;
                        sp:predicate my:age ;
                        sp:subject spin:_this
                      ] [ a       sp:Filter ;
                        sp:expression
                                [ sp:arg1 sp:_age ;
                                  sp:arg2 18 ;
                                  a sp:lt
                                ]
                      ])
]
```

Logic Programming Using RDF Data

Making Predicate Logic a Practical Tool in Programming

Logic programming = programming paradigm based on formal logic and theorem proving

→ Prolog language used in Artificial Intelligence

Problem: predicate logic is not decidable and not efficient

- There is no effective method to answer whether an arbitrary formula is logically valid

Solution: restriction to a reasonable subset of predicate logic

- Balancing between expressiveness and computational complexity (remember OWL Full vs. OWL DL)

→ Horn logic

Horn Logic & Logic Programming

A rule (clause) has the form: A1, . . ., An → B

- **A1, . . ., An** (body) is a conjunction of atomic formulas
- **B** (head) is an atomic formula

There are 2 ways of reading a rule:

- Deductive rules*:* If **A1,…, An** are known to be true, then **B** is also true
- Reactive (procedural) rules: If the conditions **A1,…, An** are true, then carry out the action **B**

Examples of Rules

- **male(X), parent(P,X), parent(P,Y), notSame(X,Y) → brother(X,Y)**
- **female(X), parent(P,X), parent(P,Y), notSame(X,Y) → sister(X,Y)**
- **brother(X,P), parent(P,Y) → uncle(X,Y)**
- **mother(X,P), parent(P,Y) → grandmother(X,Y)**
- **parent(X,Y) → ancestor(X,Y)**
- **ancestor(X,P), parent(P,Y) → ancestor(X,Y)**

Facts (rules without a body)

- → **male(John)**
- → **male(Bill)**
- → **female(Mary)**
- → **female(Jane)**
- → **parent(John,Mary)**
- → **parent(John, Bill)**

…

Queries / Goals (as rule bodies)

- **parent(John, X), female(X) →**
- **grandmother(X,Y) →**

A query is proved (by Prolog) by deriving a conflict from it (proof by contradiction)

- Solutions: value substitutions for variables

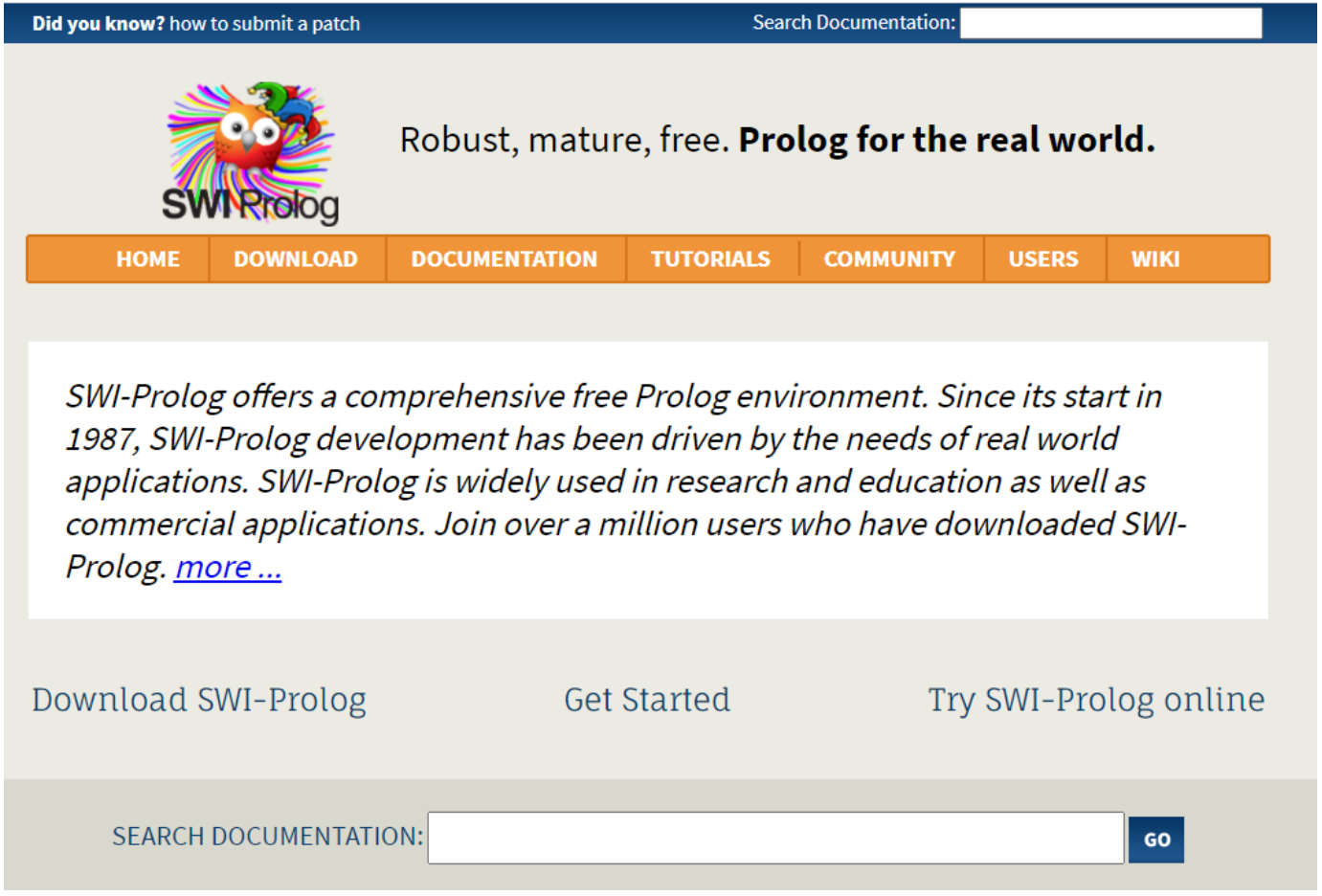
Query		**Solution/Answer**
- ***parent(John, X), female(X) ?***	->	*X=Mary;*
- ***grandmother(X,Y) ?***	->	*X=Alice, Y=Jill;* *X=George, Y=Susan;*

RDF properties and graphs are based on binary predicates

-> RDF graph = logic program!

Reasoning can be added easily on top of RDF graphs

https://www.swi-prolog.org/



Application example: recommendation of similar items in MuseumFinland

(<) **Pullonsuojus, 2 kpl:istuva koira** (> Ripustin:henkari, 'Finn Lassie')

Pullonsuojus, 2 kpl:istuva koira

Materiaali: viinapullo: lasi, pulonsuojus: lanka
Valmistaja: Karhulan lasitehdas, Tapio Wirkkala
Valmistusaika: 1962, 1970-l. n.
Valmistustekniikka: viinapullo: tehdasvalmisteinen, pulonsuojus: käsityötä
Käyttäjä: Eero Kallio
Käyttöpaikka: Etelä-Suomen lääni, Suomi
Asiasana: ALKOHOLIJUOMAT, ELÄINHAHMOT, KORISTE-ESINEET
Mitat: pullon pohjan: halkaisija 6,5cm, korkeus 22,5cm, pullonsuojuksen: korkeus 29,0cm
Museokokoelma: LAHDEN HISTORIALLINEN MUSEO
Vastuumuseo: LAHDEN KAUPUNGINMUSEO
Asiasanasto: Lahden kaupunginmuseon sanasto
Esineen numero: LKM:LHM:LHM:ES:95073:154
ID: 95073154
Viinapullo: Alkon Koskenkorvapullo. Lieriömäinen, loivat hartiat. Korkki ja etiketti puuttuvat. Pulonsuojus: istuvan koiran muotoinen pullonsuojus. Muodostuu kahdesta osasta: koiran vartalosta ja päästä. Koiran vartaloon on ommeltu viisi lankatupsua (jalat ja häntä), ylhäällä lankakiristys. Koiran pää on virkattu talouspaperirullasta leikatun lieriön ympärille. Kasvoissa mustat napit silminä, erillinen pieni kuono ja kolme lankatupsua (posket ja päälaella oleva otsatukka).

Esinetyyppi:

Sama käyttäjä

Eero Kallio:

- Keräilykortti, 14 kpl:tuotemainoskortti, erilaisia
- Kulho, 4 kpl:jälkiruokakulho
- Päähine, miehen:turkislakki, 'suikka'
- Taskuliina, miehen:taskuliinan korvike
- Jalkineet, miehen:koripallokengät

Samaan aiheeseen liittyviä esineitä

alkoholijuoma:

- kanisteri:taskumatti
- kanisteri:taskumatti
- kanisteri:taskumatti
- viinipullo:lasipullo
- pullo:lasipullo

eläimet:

- kuvakirja :kuvakirja, kangasta
- helistin :purulelu
- muovikarhu:vinkuva karhulelu
- säästölipas:vanerilipas
- malja:puuvati

Rule Interchange Format RIF

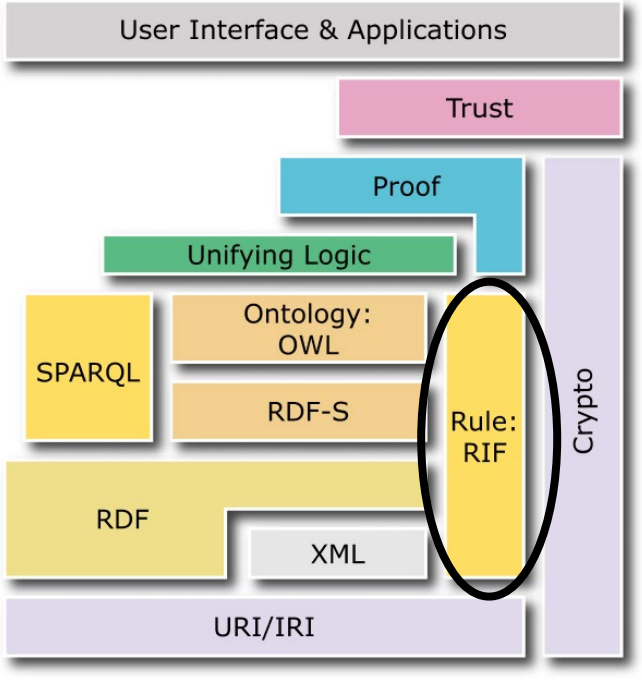
Rule Interchange Format RIF

Goals

- Rule exchange format between different rule systems
- It defines:
  - *First, a shared Core for rule systems*
  - *Then, application-specific extensions (dialects)*
- This way systems can understand each other's operation logic

Based on earlier RuleML

W3C recommendation on 5.2.2013



RIF dialects

RIF Core

- Common core of all RIF dialects
- Essentially function-free Horn logic (Datalog)
- Syntactic extensions
  - *frames (syntactic sugar), IRIs, XML datatypes, built-ins (e.g., for numeric comparison)*

RIF Basic Logic Dialect (BLD)

- Essentially Horn logic with equality, based on RIF Core
- Compatibility with RDF and OWL (RL)

RIF Production Rule Dialect (PRD)

- Reactive rules with procedural attachment
- Then part (head) of the rule contains actions

RIF example

```
Document(
  Prefix(rdf <http://www.w3.org/1999/02/22-rdf-syntax-ns#>)
  Prefix(rdfs <http://www.w3.org/2000/01/rdf-schema#>)
  Prefix(imdbrel <http://example.com/imdbrelations#>)
  Prefix(dbpedia <http://dbpedia.org/ontology>)

  Group(
    Forall ?Actor ?Film ?Role (
      If   And(rdf:type(?Actor imdbrel:Actor)
               rdf:type(?Film imdbrel:Film)
               rdf:type(?Role imdbrel:Character)
               imdbrel:playsRole(?Actor ?Role)
               imdbrel:roleInilm(?Role ?Film))
      Then dbpedia:starring(?Film ?Actor)
    )
  )
)
```

https://www.w3.org/TR/rif-overview/

W3C Working Group Note

RIF Overview (Second Edition)

W3C Working Group Note 5 February 2013

This version:
http://www.w3.org/TR/2013/NOTE-rif-overview-20130205/
Latest version:
http://www.w3.org/TR/rif-overview/
Previous version:
http://www.w3.org/TR/2012/NOTE-rif-overview-20121211/
Editors:
Michael Kifer, State University of New York at Stony Brook
Harold Boley, National Research Council Canada

A color-coded version of this document showing changes made since the previous version is also available.

This document is also available in these non-normative formats: PDF version.

Abstract

This document is an overview of the Rule Interchange Format (RIF). It provides a high-level explanation of RIF concepts and architecture as well as a general survey of RIF documents.

Status of this Document

May Be Superseded

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at http://www.w3.org/TR/.

Set of Documents

This document is being published as one of a set of 13 documents:

1. RIF Overview (Second Edition) (this document)
2. RIF Use Cases and Requirements (Second Edition)
3. RIF Core Dialect (Second Edition)
4. RIF Basic Logic Dialect (Second Edition)
5. RIF Production Rule Dialect (Second Edition)
6. RIF Framework for Logic Dialects (Second Edition)
7. RIF Datatypes and Built-Ins 1.0 (Second Edition)
8. RIF RDF and OWL Compatibility (Second Edition)
9. OWL 2 RL in RIF (Second Edition)
10. RIF Combination with XML data (Second Edition)
11. RIF In RDF (Second Edition)

RIF Summary

- **RIF is an exchange language of rules between systems**
- **Not a single one-fits-all language**
  - There are too many approaches around
- **Core + extensions (dialects)**

Semantic Web Rule Language SWRL

Aalto University
School of Science

Department of
Computer Science

SeCo

Semantic Web Rule Language SWRL

- Goal: build a bridge between OWL and Horn logic
  - SWRL = combination of function-free Horn logic and OWL DL
- Rule form: A1, . . ., An → B1,...,Bm
  - *Atom forms: C(x), P(x,y), sameAs(x,y), differentFrom(x,y)*
    - C(x): OWL description
    - P: OWL property
    - x and y: individuals, variables, or data values
- Main difficulty: restrictions for Ai and Bj are needed for decidability
  - *A prominent solution: DL-safe rules*
    - Every variable must appear in a non-description logic atom in the rule body (P(x,y) in Ai)
- OWL RL = low-end solution, SWRL high-end solution in integrating rules and DLs

SWRL Example

OWL cannot express the axiom "a person whose parents are married, is a child of married parents"

- SWRL rule expressed in OWL Functional-style syntax (can also be expressed in other OWL/RDF syntaxes and RuleML):

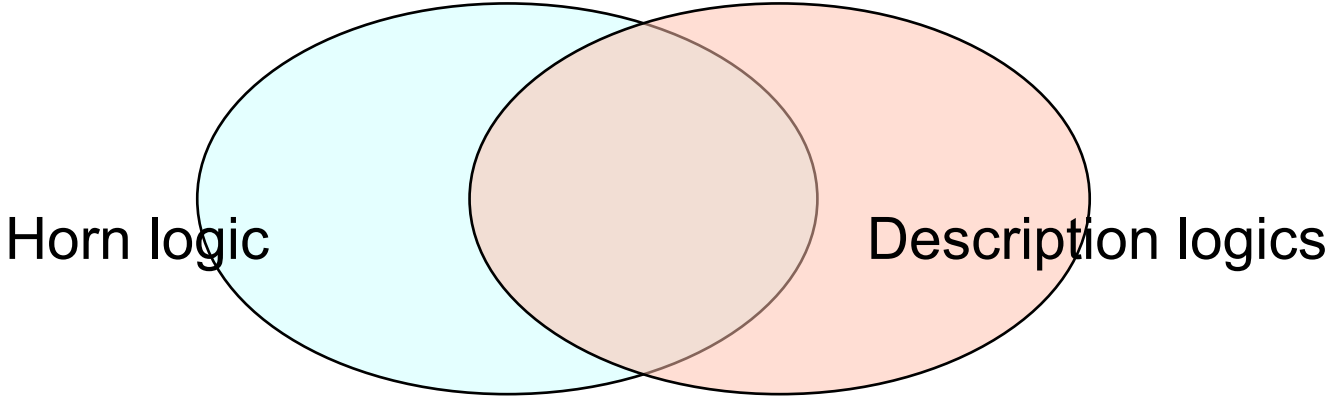
```
Prefix(var:=<urn:swrl#>)

Declaration( Class( :ChildOfMarriedParents ) )
SubClassOf( :ChildOfMarriedParents :Person )

DLSafeRule(
    Body(
        ClassAtom( :Person Variable(var:x))
        ObjectPropertyAtom( :hasParent Variable(var:x) Variable(var:y) )
        ObjectPropertyAtom( :hasParent Variable(var:x) Variable(var:z) )
        ObjectPropertyAtom( :hasSpouse Variable(var:y) Variable(var:z) )
    )
    Head(
        ClassAtom( :ChildOfMarriedParents Variable(var:x) )
    )
)
```

(Kuba, 2012)

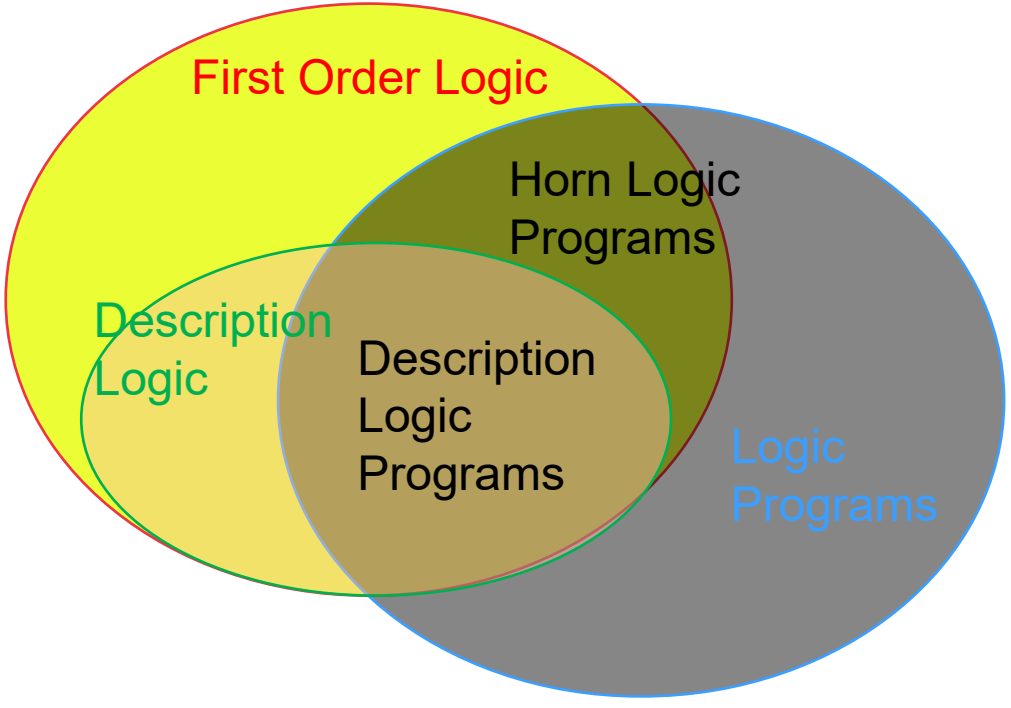
Ontologies vs. Logical Rules

Horn logic vs. description logics

- E.g., how to represent rules in description logics?
- E.g., how to represent cardinality constraints in Horn logic?



Logics of the Semantic Web



HLP = FOL & LP
DLP = DL & HLP

(Antoniou, van Harmelen, 2007)

Description Logic Programs

- Description Logic Programs (DLP) are the intersection of Horn logic and description logic
- DLP allows to combine advantages of both approaches, e.g.:
  - *A modeler may take a DL view, but*
  - *The implementation may be based on rule technology*
- -> OWL RL

Keys Challenge in Combining Logic Programming and OWL

Unique Names Assumption UNA

- Resources are different/same if they have different/same identifiers
- UNA is made in logic programming & databases but not in OWL
- UNA makes sense (only) if we know the unique identifiers

Closed World Assumption CWA

- If a fact cannot be deduced true it is assumed to be false
  - -> non-monotonic logics!
- CWA made in logic programming & databases but not in logic
- CWA makes sense (only) if our knowledge is complete

An Interoperability Problem

Logic programming & databases usually assume

- UNA + CWA

Description logics & theorem proving do not assume

- UNA + CWA

Result: different conclusions are drawn from same premises

- Interoperability is lost

Nonmonotonic Logic and Rules

Non-monotonic Reasoning

- **Our available knowledge is often *incomplete***
- **In spite of this, we need to make *plausible conclusions***
- **With new knowledge later, conclusions may become false**
  - The amount of true facts is not increasing *monotonically*
- **Problem: predicate logic is monotonic**
  - Facts can only be added not removed from the knowledge base!

Closed World Assumption Leads to Non-monotonic Reasoning

- *CWA leads to nonmonotonic behavior:*
  - *What is not known is assumed to be false*
  - *Addition of true facts may lead to removing assumptions*
- *Logic programming using Prolog is based on CWA and is using non-monotonic reasoning*

Example: Default Reasoning

- It makes sense to assume that birds can fly:
  - *Bird(X) => Flies(X)*
- If we learn that Bird(Tweety):
  - *=> Flies(Tweety)*
- However, there are exceptions to this rule
  - *Benguin(X) => -Flies(X)*
- If we later learn Benguin(Tweety):
  - *=> -Flies(Tweety)*

Contradiction: Does Tweety fly or not?

Challenge and a Solution Approach

Predicate logic systems, such as description logics (OWL) cannot deal with non-monotonic reasoning

A possible solution: select facts by prioritizing rules

- **Based on higher authority**
  - – E.g. in law, federal law preempts state law
  - – E.g., higher management has more authority than lower
- **Based on rule recency or specificity**
  - – E.g. exceptions override more general rules

New kind of non-monotonic logic systems are needed for this

Summary: Ontology and Rule Languages

The semantics of the Semantic Web is based on different subsets of the first order predicate logic

- Ontologies and OWL are based formal description logics
- Rules are based on Horn logic

Challenges of interoperability and standardization work

- Variety of logic rule systems
- UNA and CWA assumptions

Practical application of rules is possible in many ways

- SPARQL based rules
- Simple rules in OWL
- Logic programming using RDF data