



WSML Deliverable
D16.3 v1.0
**WSML ABSTRACT SYNTAX AND
SEMANTICS**

WSML Final Draft – August 8, 2008

Author:

Jos de Bruijn

Editor:

Jos de Bruijn

Final version:

<http://www.wsmo.org/TR/d16/d16.3/v0.3/20080808/>

Latest version:

<http://www.wsmo.org/TR/d16/d16.3/v1.0/>

Previous version:

<http://www.wsmo.org/TR/d16/d16.3/v1.0/20080724/>

**This document is part of the specification of the
Web Service Modeling Language (WSML), version 1.0.**

The specification of WSML 1.0 consists of the following
four documents.

WSML Language Reference

(<http://www.wsmo.org/TR/d16/d16.1/v1.0/>)

WSML Abstract Syntax and Semantics

(<http://www.wsmo.org/TR/d16/d16.3/v1.0/>)

WSML/XML (<http://www.wsmo.org/TR/d36/v1.0/>)

WSML/RDF (<http://www.wsmo.org/TR/d32/v1.0/>)



Abstract

This document presents the abstract syntax as well as the semantics of WSML. The abstract syntax closely follows the conceptual model of WSMO version 1.3, and only diverges where necessary. The document also features a mapping from the abstract to the surface syntax defined in deliverable 16.1, the WSML language reference.

The semantics of WSML ontologies is defined in terms of a model theory. The semantics of the combination of WSML ontologies with RDF graphs and OWL DL ontologies is obtained by combining the respective model theories. The notion of entailment for ontologies is used in the definition of nonfunctional property and choreography semantics.



Contents

1	WSML Abstract Syntax	4
1.1	WSML Identifiers	4
1.2	WSML Variant	5
1.3	WSML Annotation	6
1.4	WSML Nonfunctional Property	6
1.5	WSML Description	6
1.6	WSML Ontology	8
1.7	WSML Concept	8
1.8	WSML Attribute Definition	9
1.9	WSML Relation	9
1.10	WSML Parameter Definition	10
1.11	WSML Instance	10
1.12	WSML Relation Instance	10
1.13	WSML Axiom	11
1.14	WSML Web Service	11
1.15	WSML Goal	11
1.16	WSML Capability	12
1.17	WSML Interface	12
1.18	WSML Choreography	13
1.19	WSML State Signature	13
1.20	WSML Transition Rule	14
1.21	WSML Mediators	14
1.22	WSML Logical Expressions	16
2	Mapping to the Surface Syntax	18
2.1	Identifiers	18
2.2	Conceptual Syntax	18
2.3	Logical Expression Syntax	23
3	WSML Semantics	25
3.1	Ontology Semantics	25
3.1.1	WSML Variant	26
3.1.2	The Basic WSML Model Theory	26
3.1.3	WSML-DL Extensions	33
3.1.4	Stable Models for Core, Flight, and Rule	35
3.1.5	WSML Full	38
3.2	Combination with RDFS and OWL DL	39
3.2.1	Combination with RDFS	39
3.2.2	Combination with OWL DL	41
3.2.3	Satisfiability and Entailment of Combinations	42
3.3	Nonfunctional Property Semantics	46
3.4	Choreography Semantics	46
A	The Description Logic <i>SHIQ(D)</i>	49
B	WSML Layering	50



1 WSML Abstract Syntax

We present the abstract syntax of WSML. The purpose of this abstract syntax is to facilitate the direct definition of semantics of the different WSML elements, and the mapping to a different surface syntaxes.

The mapping to the surface syntax is presented in Chapter 2. The mapping to the XML syntax is presented in [8].

1.1 WSML Identifiers

A WSML vocabulary consists of the following symbols is a set of identifiers. WSML distinguishes between the following kinds of identifiers:

IRI With an *IRI* we mean any Unicode character sequence which represents a valid absolute IRI [7]. With **IRI** we denote the set of all IRIs.

In the remainder, \top and \perp are short for the IRIs `http://www.wsmo.org/wsml/wsml-syntax#true` and `http://www.wsmo.org/wsml/wsml-syntax#false`, respectively; these IRIs denote universal truths and universal falsehoods, respectively.

Anonymous identifiers `_#`, `_#1`, `_#2`, ... are *anonymous identifiers*.

Data values A *data value* is either an *elementary data value*, which is an integer, decimal, or a string (corresponding to the lexical spaces of the integer, decimal, and string datatypes defined in [3]), or a constructed data value: given an n -ary datatype wrapper w and d_1, \dots, d_n elementary data values, then $w(d_1, \dots, d_n)$ is a constructed data value.

An *identifier* is a data value, an anonymous identifier, or an IRI. We denote the set of all identifiers, which constitutes a WSML vocabulary, with **anyId**.

We denote the set of all anonymous identifiers, and IRIs with **Id** \subseteq **anyId**. The following sets of identifiers are subsets of **IRI**:

- variant identifiers,
- ontology identifiers,
- goal identifiers,
- Web service identifiers,
- ooMediator identifiers,
- ggMediator identifiers,
- wgMediator identifiers,
- wwMediator identifiers,
- datatype wrapper identifiers,



- built-in predicate identifiers,
- capability identifiers,
- interface identifiers,
- choreography identifiers,
- orchestration identifiers,
- state signature identifiers, and
- grounding identifiers,

which are mutually disjoint, and each datatype wrapper and built-in predicates identifier has an associated arity n , which is a positive integer.

The following sets of identifiers are subsets of **Id**:

- annotation property identifiers,
- nonfunctional property identifiers,
- concept identifiers,
- attribute identifiers,
- relation identifiers,
- function identifiers,
- instance identifiers,
- relation instance identifiers, and
- axiom identifiers.

Finally, the set of capability identifiers is disjoint from the set of concept identifiers.

A *variable symbol* is an alphanumeric string preceded by a question mark ‘?’.

Any WSML vocabulary conforms with WSML-Full, WSML-Rule, and WSML-Flight. If the mentioned sets of identifiers are mutually disjoint, then the WSML vocabulary conforms with WSML-DL and WSML-Core.

1.2 WSML Variant

A *WSML variant* is a sub language of WSML which may pose syntactic and semantic restrictions on the language. A *WSML variant identifier* is a IRI. The currently defined WSML variants are:



<i>Variant</i>	<i>Identifier</i>
WSML Full	http://www.wsmo.org/wsml/wsml-syntax/wsml-full
WSML DL	http://www.wsmo.org/wsml/wsml-syntax/wsml-dl
WSML Rule	http://www.wsmo.org/wsml/wsml-syntax/wsml-rule
WSML Flight	http://www.wsmo.org/wsml/wsml-syntax/wsml-flight
WSML Core	http://www.wsmo.org/wsml/wsml-syntax/wsml-core

WSML Full includes all of WSML. WSML DL is a subset which can be processed using Description Logic reasoners, and is compatible with OWL DL. WSML Rule is a subset which can be processed using (nonmonotonic) rule reasoners, and is compatible with the Stable Model Semantics for logic programs. WSML Flight is a decidable subset of which will Rule. WSML Core is a subset of the DL and Flight variants, and can thus be processed by both Description Logic and rule reasoners.

1.3 WSML Annotation

A *WSML annotation* is a 2-tuple $\langle \text{name}, \text{value} \rangle_{ann}$, where

- name is an annotation property identifier and
- value is an identifier in **anyId**.

1.4 WSML Nonfunctional Property

A *WSML nonfunctional property* is a 3-tuple $\langle \text{name}, \text{value}, \mathbf{logExp} \rangle_{nfp}$, where

- name is a nonfunctional property identifier,
- value is an identifier in **anyId** or a variable symbol, and
- **logExp** is a set of WSML logical expressions.

Conformance A WSML nonfunctional property $\langle \text{name}, \text{value}, \mathbf{logExp} \rangle_{nfp}$ conforms with WSML Core, Flight, and Rule if for every logical expression in **logExp** holds that it does not contain the symbols \neg , \supset , and \equiv .¹

A WSML nonfunctional property $\langle \text{name}, \text{value}, \mathbf{logExp} \rangle_{nfp}$ conforms with WSML DL if for every logical expression in **logExp** holds that it contains one free variable and its variable graph is tree-shaped.²

Any nonfunctional property conforms with WSML Full.

1.5 WSML Description

A *WSML description* is a 5-tuple $\langle \text{varID}, \mathbf{O}, \mathbf{G}, \mathbf{WS}, \mathbf{M} \rangle_{description}$, where

¹In fact, every such logical expression is a (FOL) query.

²In fact, every such logical expression is a tree-shaped query.



- varID is a *WSML variant identifier*,
- **O** is a set of *Ontologies*,
- **G** is a set of *WSML goals*,
- **WS** is a set of *WSML Web services*, and
- **M** is a set of *WSML mediators*.

Ontologies are either RDF Schema, OWL DL or Full, or WSML ontologies. The latter are defined in Section 1.6. The abstract syntax of RDF Schema and OWL Full ontologies is that of RDF [16]. The abstract syntax of OWL DL is defined in [18]. Extensions may allow other kinds of ontologies, e.g. OWL 1.1 (<http://www.webont.org/owl/1.1/>) or the upcoming RIF standard (<http://www.w3.org/2005/rules/>).

Conformance A WSML description $\langle \text{varID}, \mathbf{O}, \mathbf{G}, \mathbf{WS}, \mathbf{M} \rangle_{\text{description}}$ is *conforming* if its vocabulary conforms with the variant denoted by varID, **O**, **G**, **WS** and **M** conform with the variant denoted by varID, and the variant of every imported ontology and the description is lower than or equal to *variant*.

A set of ontologies **O** conforms with a variant *variant* if for every ontology in $o \in \mathbf{O}$ holds that

- if O is a WSML ontology, then the attribute definitions and logical expressions in **O** conform with the variant denoted by varID, and **O** does not contain any relations or relation instances if *variant* is WSML-Core or WSML-DL,
- if O is an RDF graph, then *variant* is not WSML-Core or WSML-DL,
- if O is an OWL DL ontology, then *variant* is WSML-DL or WSML-Full, and
- if O is an OWL Full ontology, then *variant* is WSML-Full,

and the vocabularies of the ontologies in **O** corresponds as follows:

- all OWL DL ontologies in **O** share the same vocabulary V and the following correspondences hold between **anyId** and V if *variant* is WSML DL:
 - the sets of ontology identifiers and **anyId** and V correspond,
 - the set of datatype wrapper identifiers in **anyId** corresponds with the set of datatype identifiers in V ,
 - the set of annotation property identifiers in **anyId** which are IRIs corresponds with the sets of ontology and annotation property identifiers in V ,
 - the set of concept identifiers in **anyId** which are IRIs corresponds with the union of the sets of class and datatype identifiers in V ,
 - the set of attribute identifiers in **anyId** which are IRIs corresponds with the union of the sets of data valued and individual valued property identifiers in V , and



- the set of instance identifiers in **anyId** which are IRIs corresponds with the set of individual identifiers in V .

A set of goals **G**, Web services **WS**, or mediators **M** conforms with the variant *variant* if

- every nonfunctional property in **G**, **WS**, and **M** conforms with *variant*, and
- every choreography in **G**, **WS**, and **M** conforms with *variant*.

Extensions may pose additional restrictions on the shape of logical expressions in **G** and **WS**. **TODO: conformance for choreographies, especially conditions on the logical expressions in transition rules**

1.6 WSML Ontology

A *WSML ontology* is a 9-tuple $\langle \text{name}, \mathbf{ann}, \mathbf{ontID}, \mathbf{medID}, \mathbf{concept}, \mathbf{relation}, \mathbf{instance}, \mathbf{relInstance}, \mathbf{axiom} \rangle_{ontology}$, where

- **name** is an ontology identifier or the symbol `nil`,
- **ann** is a set of *WSML annotations*,
- **ontID** is a set of ontology identifiers (imported ontologies),
- **medID** is a set of ooMediator identifiers,
- **concept** is a set of *WSML concepts*,
- **relation** is a set of *WSML relations*,
- **instance** is a set of *WSML instances*,
- **relInstance** is a set of *WSML relation instances*, and
- **axiom** is a set of *WSML axioms*.

1.7 WSML Concept

A *WSML concept* is a 4-tuple $\langle \text{name}, \mathbf{ann}, \mathbf{conceptID}, \mathbf{attribute} \rangle_{concept}$, where

- **name** is a concept identifier or the symbol `nil`,
- **ann** is a set of *WSML annotations*,
- **conceptID** is a set of (super-)concept identifiers, and
- **attribute** is a set of *WSML attribute definitions*.



1.8 WSML Attribute Definition

A *WSML attribute definition* is a 7-tuple $\langle \text{name}, \mathbf{ann}, \text{defType}, \mathbf{feature}, \text{minCard}, \text{maxCard}, \mathbf{rangeID} \rangle_{ad}$, where

- name is an attribute identifier or the symbol nil,
- **ann** is a set of *WSML annotations*,
- defType is either *impliesType* or *ofType*,
- **feature** is a set of *WSML attribute features*, where a WSML attribute feature is one of the following:
 - *transitive*,
 - *reflexive*,
 - *symmetric*,
 - *subAttribute attID*, where attID is an attribute identifier, or
 - *inverse attID*, where attID is an attribute identifier.
- minCard is a nonnegative integer,
- maxCard is a nonnegative integer or the symbol nil,
- **rangeID** is a set of concept identifiers.

Any WSML attribute definition *conforms with* WSML-Full, WSML-Rule, and WSML-Flight. If for a WSML attribute definition $\text{attDef} = \langle \text{name}, \mathbf{ann}, \text{defType}, \mathbf{feature}, \text{minCard}, \text{maxCard}, \mathbf{rangeID} \rangle$ holds that $\text{defType} = \text{impliesType}$, $\mathbf{feature} = \emptyset$, $\text{minCard} = 0$, and $\text{maxCard} = \text{nil}$, then attDef conforms with WSML-DL and WSML-Core.

1.9 WSML Relation

A *WSML relation* is a 4-tuple $\langle \text{name}, \mathbf{ann}, \mathbf{relationID}, \overline{\text{parameter}} \rangle_{relation}$, where

- name is a relation identifier or the symbol nil,
- **ann** is a set of *WSML annotations*,
- **relationID** is a set of (super-)relation identifiers, and
- $\overline{\text{parameter}}$ is a vector of *WSML parameter definitions*.



1.10 WSML Parameter Definition

A *WSML parameter definition* is a 3-tuple $\langle \mathbf{ann}, \text{defType}, \text{domainID} \rangle_{pd}$, where

- \mathbf{ann} is a set of *WSML annotations*,
- defType is either *impliesType* or *ofType*, and
- domainID is a concept identifier.

1.11 WSML Instance

A *WSML instance* is a 4-tuple $\langle \text{name}, \mathbf{ann}, \text{conceptID}, \text{attVal} \rangle_{instance}$, where

- name is an instance identifier or the symbol *nil*,
- \mathbf{ann} is a set of *WSML annotations*,
- conceptID is a set of concept identifiers, and
- attVal is a set of *WSML attribute value pairs*, where a *WSML attribute value pair* is a 3-tuple $\langle \text{name}, \mathbf{ann}, \text{valueID} \rangle_{av}$, where
 - name is an attribute identifier or the symbol *nil*,
 - \mathbf{ann} is a set of *WSML annotations*, and
 - valueID is a set of instance and data value identifiers.

1.12 WSML Relation Instance

A *WSML relation instance* is a 4-tuple $\langle \text{name}, \mathbf{ann}, \text{relationID}, \overline{\text{parVal}} \rangle_{ri}$, where

- name is a relation instance identifier or the symbol *nil*,
- \mathbf{ann} is a set of *WSML annotations*,
- relationID is a relation identifier, and
- $\overline{\text{parVal}}$ is a vector of *WSML parameter values*, where a *WSML parameter value* is a 2-tuple $\langle \mathbf{ann}, \text{valueID} \rangle_{pv}$, where
 - \mathbf{ann} is a set of *WSML annotations*, and
 - valueID is an instance and data value identifier.



1.13 WSML Axiom

A *WSML axiom* is a 3-tuple $\langle \text{name}, \mathbf{ann}, \mathbf{logExp} \rangle_{\text{axiom}}$, where

- name is an axiom identifier or the symbol nil,
- **ann** is a set of *WSML annotations*, and
- **logExp** is a set of *WSML logical expressions*.

1.14 WSML Web Service

A *WSML Web service* is a 7-tuple $\langle \text{name}, \mathbf{ann}, \mathbf{ontID}, \mathbf{medID}, \mathbf{nfp}, \text{capability}, \mathbf{interface} \rangle_{\text{ws}}$, where

- name is a Web service identifier or the symbol nil,
- **ann** is a set of *WSML annotations*,
- **ontID** is a set of ontology identifiers,
- **medID** is a set of ooMediator and wwMediator identifiers,
- **nfp** is a set of *WSML nonfunctional properties*,
- capability is a *WSML capability* or a concept identifier,
- **interface** is a set of *WSML interfaces*, and

the sets of ontology identifiers in capability and every interface in **interface** are supersets of **ontID**.

The latter restriction ensures that all ontologies imported in a Web service are also imported in the contained capability and interfaces.

In case capability is a concept identifier, it is assumed to identify a concept in a task ontology, which is imported by the Web service.

1.15 WSML Goal

A *WSML goal* is a 7-tuple $\langle \text{name}, \mathbf{ann}, \mathbf{ontID}, \mathbf{medID}, \mathbf{nfp}, \text{capability}, \mathbf{interface} \rangle_{\text{goal}}$, where

- name is an goal identifier or the symbol nil,
- **ann** is a set of *WSML annotations*,
- **ontID** is a set of ontology identifiers,
- **medID** is a set of ooMediator and ggMediator identifiers,
- **nfp** is a set of *WSML nonfunctional properties*,



- capability is a *WSML capability* or a concept identifier,
- **interface** is a set of *WSML interfaces*, and

the sets of ontology identifiers in capability and every interface in **interface** are supersets of **ontID**.

1.16 WSML Capability

A *WSML capability* is a 10-tuple $\langle \text{name}, \mathbf{ann}, \mathbf{ontID}, \mathbf{medID}, \mathbf{nfp}, \mathbf{sharedVar}, \mathbf{pre}, \mathbf{post}, \mathbf{ass}, \mathbf{eff} \rangle_{\text{capability}}$, where

- name is a capability identifier or the symbol nil,
- **ann** is a set of *WSML annotations*,
- **ontID** is a set of ontology identifiers,
- **medID** is a set of ooMediator and wgMediator identifiers,
- **nfp** is a set of *WSML nonfunctional properties*,
- **sharedVar** is a set of *WSML variables*, i.e. alphanumeric strings preceded with a question mark '?',
- **pre** is a set of *WSML axioms*,
- **post** is a set of *WSML axioms*,
- **ass** is a set of *WSML axioms*, and
- **eff** is a set of *WSML axioms*.

1.17 WSML Interface

A *WSML interface* is a 7-tuple $\langle \text{name}, \mathbf{ann}, \mathbf{ontID}, \mathbf{medID}, \mathbf{nfp}, \mathbf{chor}, \mathbf{orchID} \rangle_{\text{interface}}$, where

- name is an interface identifier or the symbol nil,
- **ann** is a set of *WSML annotations*,
- **ontID** is a set of ontology identifiers,
- **medID** is a set of ooMediator identifiers,
- **nfp** is a set of *WSML nonfunctional properties*,
- **chor** is a set of *WSML choreographies*,
- **orchID** is a set of orchestration identifiers, and

the set of ontology identifiers in every chor in **chor** is a superset of **ontID**. Extensions may allow other kinds of choreographies, e.g. based on UML state charts.



1.18 WSML Choreography

A *WSML choreography* is a 6-tuple $\langle \text{name}, \mathbf{ann}, \mathbf{ontID}, \mathbf{medID}, \text{signature}, \mathbf{rule} \rangle_{\text{chor}}$, where

- **name** is a choreography identifier or the symbol *nil*,
- **ann** is a set of *WSML annotations*,
- **ontID** is a set of ontology identifiers,
- **medID** is a set of ooMediator identifiers,
- **signature** is a *WSML state signature*,
- **rule** is a set of *WSML transition rules*, and

the set of ontology identifiers in **signature** is a superset of **ontID**.

Conformance A WSML choreography $\langle \text{name}, \mathbf{ann}, \mathbf{ontID}, \mathbf{medID}, \text{signature}, \mathbf{rule} \rangle_{\text{chor}}$ conforms with WSML Core (resp., Flight or Rule) if for every logical expression in every rule in **rule** holds that it does not contain the symbols \neg , \supset , and \equiv .³

A WSML choreography $\langle \text{name}, \mathbf{ann}, \mathbf{ontID}, \mathbf{medID}, \text{signature}, \mathbf{rule} \rangle_{\text{chor}}$ conforms with WSML DL if for every logical expression in every rule in **rule** holds that it contains one free variable and its variable graph is tree-shaped.⁴

Any WSML choreography conforms with WSML Full.

1.19 WSML State Signature

A *WSML state signature* is a 5-tuple $\langle \text{name}, \mathbf{ann}, \mathbf{ontID}, \mathbf{medID}, \mathbf{mode} \rangle_{\text{ss}}$, where

- **name** is a state signature identifier or the symbol *nil*,
- **ann** is a set of *WSML annotations*,
- **ontID** is a set of ontology identifiers,
- **medID** is a set of ooMediator identifiers, and
- **mode** is a set of *modes*, where a *mode* is either a *concept mode* of the form $\langle \text{type}, \text{conceptID}, \mathbf{groundingID} \rangle_{\text{cm}}$ or *relation mode* of the form $\langle \text{type}, \text{relationID}, \mathbf{groundingID} \rangle_{\text{rm}}$, where
 - **type** is one of the symbols *static*, *in*, *out*, *shared*, and *controlled*,
 - **conceptID** (resp., **relationID**) is a concept (resp., relation) identifier, and
 - **groundingID** is a set of grounding identifiers.

³In fact, every such logical expression is a (FOL) query.

⁴In fact, every such logical expression is a tree-shaped query.



1.20 WSML Transition Rule

A *WSML transition rule* is

- an *if rule* $\langle \text{logExp}, \text{rule} \rangle_{if}$,
- a *forall rule* $\langle \text{varID}, \text{logExp}, \text{rule} \rangle_{forall}$,
- a *choose rule* $\langle \text{varID}, \text{logExp}, \text{rule} \rangle_{choose}$,
- a *piped rule* $\langle \text{rule} \rangle_{piped}$,
- an *add rule* $\langle \text{fact} \rangle_{add}$, or
- a *delete rule* $\langle \text{fact} \rangle_{delete}$,

where

- logExp is a *WSML logical expression* such that the set of free variables in logExp corresponds with varID ,
- rule is a nonempty set of *WSML transition rules*,
- varID is a set of variable identifiers, and
- fact is a *WSML fact* (i.e. ground atomic formula).

1.21 WSML Mediators

A *WSML ooMediator* is a 7-tuple $\langle \text{name}, \text{ann}, \text{ontID}, \text{nfp}, \text{sourceID}, \text{targetID}, \text{serviceID} \rangle_{oom}$, where

- name is an *ooMediator identifier* or the symbol *nil*,
- ann is a set of *WSML annotations*,
- ontID is a set of *ontology identifiers*,
- nfp is a set of *WSML nonfunctional properties*,
- sourceID is a set of *ontology and ooMediator identifiers*,
- targetID is a set of *ontology, goal, Web service, and mediator identifiers*, and
- serviceID is a set of *goal, Web service, and wwMediator identifiers*.

A *WSML ggMediator* is an 8-tuple $\langle \text{name}, \text{ann}, \text{ontID}, \text{medID}, \text{nfp}, \text{sourceID}, \text{targetID}, \text{serviceID} \rangle_{ggm}$, where

- name is a *ggMediator identifier* or the symbol *nil*,
- ann is a set of *WSML annotations*,
- ontID is a set of *ontology identifiers*,



- **medID** is a set of ooMediator identifiers,
- **nfp** is a set of *WSML nonfunctional properties*,
- **sourceID** is a set of goal and ggMediator identifiers,
- **targetID** is a set of goal and ggMediator identifiers, and
- **serviceID** is a set of goal, Web service, and wwMediator identifiers.

A *WSML wgMediator* is an 8-tuple $\langle \text{name, ann, ontID, medID, nfp, sourceID, targetID, serviceID} \rangle_{wgM}$, where

- name is a ggMediator identifier or the symbol nil,
- **ann** is a set of *WSML annotations*,
- **ontID** is a set of ontology identifiers,
- **medID** is a set of ooMediator identifiers,
- **nfp** is a set of *WSML nonfunctional properties*,
- **sourceID** is a set of goal, Web service, ggMediator and wgMediator identifiers,
- **targetID** is a set of goal, Web service, ggMediator and wgMediator identifiers, and
- **serviceID** is a set of goal, Web service, and wwMediator identifiers.

A *WSML wwMediator* is an 8-tuple $\langle \text{name, ann, ontID, medID, nfp, sourceID, targetID, serviceID} \rangle_{wwM}$, where

- name is a ggMediator identifier or the symbol nil,
- **ann** is a set of *WSML annotations*,
- **ontID** is a set of ontology identifiers,
- **medID** is a set of ooMediator identifiers,
- **nfp** is a set of *WSML nonfunctional properties*,
- **sourceID** is a set Web service and wwMediator identifiers,
- **targetID** is a set of Web service and wwMediator identifiers, and
- **serviceID** is a set of goal, Web service, and wwMediator identifiers.



1.22 WSML Logical Expressions

A *WSML logical expression* is a *WSML sentence*, as defined in the following. A WSML logical expression ϕ *conforms with* WSML-Full, WSML-DL, WSML-Rule, WSML-Flight, or WSML-Core, respectively, if ϕ is a WSML-Full, WSML-DL, WSML-Rule, WSML-Flight, or WSML-Core sentence, respectively.

Concrete terms are constructed using datatype wrappers and variable symbols: every variable symbol is a concrete term; if $u \in \mathbf{IRI}$ is a datatype wrapper identifier and t_1, \dots, t_n are concrete terms, with $1 \leq n$, then $u(t_1, \dots, t_n)$ is a concrete term.

Abstract terms are constructed using identifiers and variable symbols in the following way: any identifier in **anyId** and any variable symbol is an abstract term; if u is a function identifier and t_1, \dots, t_n are terms, with $1 \leq n$, then $u(t_1, \dots, t_n)$ is an abstract term.

Abstract atomic formulas (atoms) are \top, \perp or are constructed from terms and relation identifiers in the following way: any relation identifier p is an abstract atomic formula and if t_1, \dots, t_n are terms, with $1 \leq n$, then $p(t_1, \dots, t_n)$ is an abstract atomic formula. *Concrete atomic formulas* (atoms) are constructed from concrete terms and built-in predicate identifiers: any built-in predicate identifier p is a concrete atomic formula and if t_1, \dots, t_n are concrete terms, with $1 \leq n$, then $p(t_1, \dots, t_n)$ is a concrete atomic formula. Atoms are either abstract or concrete atoms.

Molecules are defined as follows: if t_1, t_2, t_3 are terms, then $t_1 : t_2$, $t_1 :: t_2$ and $t_1[t_2 \times t_3]$, with $\times \in \{\text{ot}, \text{it}, \text{hv}\}$, are molecules. The symbol **ot** stands for the WSML construct *ofType*; a statement $t_1[t_2 \text{ot} t_3]$ requires all values for the attribute t_2 to be *known* to be of a member of the type t_3 ; **it** stands for the WSML construct *impliesType*; a statement $t_1[t_2 \text{it} t_3]$ implies that all values for the attribute t_2 are a member of the class t_3 ; **hv** stands for the WSML construct *hasValue*; a molecule $t_1[t_2 \text{hv} t_3]$ says that the individual t_1 has an attribute t_2 with value t_3 .

WSML formulas are inductively defined as follows, with $\phi, \psi \in \mathcal{L}$:

- atoms and molecules are formulas;
- $\sim \phi$, with $\sim \in \{\neg, \text{not}\}$, is a formula;
- $\phi \star \psi$, with $\star \in \{\wedge, \vee, \supset, \equiv\}$, is a formula; and
- $Q x(\phi)$, with $Q \in \{\forall_a, \exists_a, \forall_c, \exists_c\}$ and $x \in \mathcal{V}$, is a formula.

Additionally, no variable quantified using an abstract quantifier (\forall_a, \exists_a) may be used in a concrete atom. WSML sentences are WSML formulas with no free variables.

WSML-Full Any WSML sentence is a WSML-Full sentence. A *WSML-Full theory* is a set of WSML-Full sentences.

WSML-Rule WSML-Rule formulas are of the form

$$(\forall) b_1 \wedge \dots \wedge b_l \wedge \text{not } c_1 \wedge \dots \wedge \text{not } c_m \supset h \quad (1.1)$$

where $b_1, \dots, b_l, c_1, \dots, c_m$ are atoms or molecules, with l, m nonnegative integers, and h an abstract equality-free atom or molecule; if $h = \perp$, then we call



the rule an *integrity constraint*. Additionally, each quantifier is either abstract (\forall_a) or concrete (\forall_c). A *WSML-Rule theory* is a set of WSML-Rule sentences.

Concrete atoms in WSML-Rule correspond to the common built-in atoms in Logic Programming.

WSML-Flight A *WSML-Flight theory* is a WSML-Rule theory for which holds that, for every formula of the form (1.1), every variable occurs in a positive abstract body atom b_i , no function symbol in (1.1) is used with an arity higher than 0, and the theory is *locally stratified*.⁵⁶

WSML-DL Given an FOL formula ϕ , $\delta'(\phi)$ is obtained from ϕ by replacing atoms of the forms $A(t_1)$, $R(t_1, t_2)$, with t_1, t_2 terms, with molecules of the forms $t_1 : A, t_1[R \text{ hv } t_2]$.

Given a *SHIQ(D)* language with signature $\Sigma = \langle \mathcal{C}, \mathcal{D}, \mathcal{R}_a, \mathcal{R}_c, \mathcal{F}_a, \mathcal{F}_c \rangle$ such that \mathcal{C} is the set of concept identifiers, \mathcal{D} is the set of built-in predicates identifiers, $\mathcal{R}_a \cup \mathcal{R}_c$ is the set of attributes identifiers, \mathcal{F}_a is the set of instance identifiers, and \mathcal{F}_c corresponds to the set of data values. A WSML-DL formula is a WSML formula of the form

- $\delta'(\phi)$, where ϕ is the FOL equivalent of a *SHIQ(D)* axiom (see Appendix A),
- $a :: b$, with $a, b \in \mathcal{C}$,
- $a[s \text{ it } b]$, with $s \in \mathcal{R}_a$ and $a, b \in \mathcal{C}$,
- $a[u \text{ it } d]$, with $u \in \mathcal{R}_c$, $a \in \mathcal{C}$ and $d \in \mathcal{D}$, or
- $a[p \text{ hv } b]$, with $a \in \mathbf{Id}$ and p an annotation property identifier.

A *WSML-DL theory* is a set of WSML-DL sentences.

WSML-Core A WSML-DL formula which is also a Flight formula is a WSML-Core formula. *WSML-Core theory* is a set of WSML-Core sentences.

⁵Each atom or molecule in $gr(\Phi)$ is assigned a stratum, which is an integer. We say that $gr(\Phi)$ is stratified if there is an assignment of atoms and molecules to strata such that: if an atom or molecule p occurs positively in a rule with an atom or molecule q as its head, then p has the same or a lower stratum, and if p occurs negatively in a rule with q as its head, then p has a lower stratum than q . If $gr(\Phi)$ is stratified, then Φ is locally stratified.

⁶These conditions correspond to the usual safety condition which must hold for Datalog programs, and the usual local stratification for logic programs.



$tr(uri)$	$=$	$"uri"$	$uri \in \mathbf{IRI}$ is a IRI and uri' is obtained from uri by replacing every $"$ with \backslash
$tr(ai)$	$=$	ai	ai is an anonymous identifier
$tr(v)$	$=$	v	v is a variable symbol
$tr(int)$	$=$	int	int is an integer or decimal
$tr(str)$	$=$	$"str"$	str is a string and str' is obtained from str by replacing every $"$ with \backslash
$tr(dw(dv_1, \dots, dv_n))$	$=$	$tr(dw)(tr(dv_1), \dots, tr(dv_n))$	$dw(dv_1, \dots, dv_n)$ is a constructed data value
$tr(nil)$	$=$		the symbol nil is discarded

Table 2.1: Mapping identifiers to the surface syntax

2 Mapping to the Surface Syntax

This chapter shows the mapping from the abstract syntax to the surface syntax, which is used in the reference document.

Throughout this chapter we define the mapping function tr which takes as argument a WSML description in the abstract syntax, and returns the equivalent in the surface syntax.

2.1 Identifiers

Table 2.1 defines the mapping for identifiers.

Additionally, the following *shortcut syntax* may be used in the surface syntax:

- The shortcuts for datatype wrappers corresponding to the XML schema datatype identifiers defined in Table C.1 of deliverable 16.1 may be used.
- IRIs may be abbreviated using sQNames, using the namespace mechanism defined in Section 2.1 of deliverable 16.1.

2.2 Conceptual Syntax

Tables 2.2, 2.3, 2.4, and 2.5 define the mapping for the conceptual syntax.

Additionally, the following *shortcut syntax* may be used in the surface syntax:

- A set of WSML annotations of the form $\{\langle \text{name}, \text{value}_1 \rangle_{ann}, \dots, \langle \text{name}, \text{value}_n \rangle_{ann}\}$ may be written as $tr(\text{name}) \text{ hasValue } \{tr(\text{value}_1), \dots, tr(\text{value}_n)\}$.
- In the translation of WSML descriptions, if $varID$ denotes the variant WSML Full, then the identification of the variant $tr(varID)$ may be omitted.



$tr(\langle \text{name}_1, \text{value}_1 \rangle, \dots, \langle \text{name}_n, \text{value}_n \rangle_{\text{ann}})$	$=$	annotations $tr(\text{name}_1)$ hasValue $tr(\text{value}_1)$... $tr(\text{name}_n)$ hasValue $tr(\text{value}_n)$ endAnnotations
$tr(\langle \text{name}, \text{value}, \emptyset \rangle)$	$=$	nonFunctionalProperty $tr(\text{name})$ hasValue $tr(\text{value})$ nonFunctionalProperty $tr(\text{name})$ hasValue $tr(\text{value})$ definedBy
$tr(\langle \text{name}, \text{value}, \{\text{logExp}_1, \dots, \text{logExp}_n\} \rangle)$	$=$	$tr(\text{logExp}_1)$ $tr(\text{logExp}_n)$.
$tr_{io}(\{\text{ontID}_1, \dots, \text{ontID}_n\})$	$=$	importsOntology $\{tr(\text{ontID}_1), \dots, tr(\text{ontID}_n)\}$
sets of ontology identifiers (imported ontologies)		
$tr_{um}(\{\text{medID}_1, \dots, \text{medID}_n\})$	$=$	usesMediator $\{tr(\text{medID}_1), \dots, tr(\text{medID}_n)\}$
sets of mediator identifiers (used mediators)		
$tr(\langle \text{varID}, \{\text{O}_1, \dots, \text{O}_n\}, \{\text{G}_1, \dots, \text{G}_m\}, \{\text{WS}_1, \dots, \text{WS}_l\}, \{\text{M}_1, \dots, \text{M}_k\}_{\text{description}} \rangle)$	$=$	variant $tr(\text{varID})$ $tr(\text{O}_1)$... $tr(\text{O}_n)$ $tr(\text{G}_1)$... $tr(\text{G}_m)$ $tr(\text{WS}_1)$... $tr(\text{WS}_l)$ $tr(\text{M}_1)$... $tr(\text{M}_k)$
$tr(\langle \text{name}, \mathbf{ann}, \mathbf{ontID}, \mathbf{medID}, \{\text{concept}_1, \dots, \text{concept}_n\}, \{\text{relation}_1, \dots, \text{relation}_m\}, \{\text{instance}_1, \dots, \text{instance}_l\}, \{\text{relInstance}_1, \dots, \text{relInstance}_k\}, \{\text{axiom}_1, \dots, \text{axiom}_o\}_{\text{ontology}} \rangle)$	$=$	ontology $tr(\text{name})$ $tr(\mathbf{ann})$ $tr_{io}(\mathbf{ontID})$ $tr_{um}(\mathbf{medID})$ $tr(\text{concept}_1)$... $tr(\text{concept}_n)$ $tr(\text{relation}_1)$... $tr(\text{relation}_m)$ $tr(\text{instance}_1)$... $tr(\text{instance}_l)$ $tr(\text{relInstance}_1)$... $tr(\text{relInstance}_k)$ axiom $tr(\text{axiom}_1)$... axiom $tr(\text{axiom}_o)$
$tr(\langle \text{name}, \mathbf{ann}, \{\text{conceptID}_1, \dots, \text{conceptID}_n\}, \{\text{attribute}_1, \dots, \text{attribute}_m\}_{\text{concept}} \rangle)$	$=$	concept $tr(\text{name})$ subConceptOf{ $tr(\text{conceptID}_1), \dots, tr(\text{conceptID}_n)$ } $tr(\mathbf{ann})$ $tr(\text{attribute}_1)$... $tr(\text{attribute}_m)$
$tr(\langle \text{name}, \mathbf{ann}, \text{defType}, \{\text{feature}_1, \dots, \text{feature}_n\}, \text{minCard}, \text{maxCard}, \{\text{rangeID}_1, \dots, \text{rangeID}_m\}_{\text{ad}} \rangle)$	$=$	$tr(\text{name})$ $tr(\text{feature}_1)$... $tr(\text{feature}_n)$ $tr(\text{defType})$ $(\text{minCard} \text{maxCard}') \{tr(\text{rangeID}_1), \dots, tr(\text{rangeID}_m)\} tr(\mathbf{ann})$
$\text{maxCard}' = \text{maxCard}$ if maxCard is a nonnegative integer; $\text{maxCard} = *$, otherwise		

Table 2.2: Mapping conceptual syntax to the surface syntax (I)



$tr(\text{impliesType})$	$=$	<code>impliesType</code>
$tr(\text{ofType})$	$=$	<code>ofType</code>
$tr(\text{transitive})$	$=$	<code>transitive</code>
$tr(\text{reflexive})$	$=$	<code>reflexive</code>
$tr(\text{symmetric})$	$=$	<code>symmetric</code>
$tr(\text{inverse } attID)$	$=$	<code>inverseOf(tr(attID))</code>
$tr(\text{subAttribute } attID)$	$=$	<code>subAttributeOf(tr(attID))</code>
$tr(\langle \text{name, ann, } \{ \text{relationID}_1, \dots, \text{relationID}_n \}, \langle \text{parameter}_1, \dots, \text{parameter}_m \rangle \rangle_{\text{relation}})$	$=$	<code>relation tr(name)(tr(parameter₁), ..., tr(parameter_n))</code> <code>subRelationOf {tr(relationID₁), ..., tr(relationID_n)} tr(ann)</code>
$tr(\langle \text{ann, defType, domainID} \rangle_{pd})$	$=$	<code>tr(defType) tr(domainID) tr(ann)</code>
$tr(\langle \text{name, ann, } \{ \text{conceptID}_1, \dots, \text{conceptID}_n \}, \{ \text{attVal}_1, \dots, \text{attVal}_m \} \rangle_{\text{instance}})$	$=$	<code>instance tr(name) memberOf {tr(conceptID₁), ..., tr(conceptID_n)}</code> <code>tr(ann) tr(attVal₁) ... tr(attVal_m)</code>
$tr(\langle \text{name, ann, } \{ \text{valueID}_1, \dots, \text{valueID}_n \} \rangle_{\text{ava}})$	$=$	<code>tr(name) hasValue {tr(valueID₁), ..., tr(valueID_n)}</code> <code>tr(ann)</code>
$tr(\langle \langle \text{name, ann, relationID, } \{ \text{parVal}_1, \dots, \text{parVal}_n \} \rangle_{ri})$	$=$	<code>relationInstance tr(name) tr(relationID)</code> <code>(tr(parVal₁), ..., tr(parVal_n)) tr(ann)</code>
$tr(\langle \text{ann, valueID} \rangle_{pv})$	$=$	<code>tr(valueID) tr(ann)</code>
$tr(\langle \text{name, ann, } \emptyset \rangle_{\text{axiom}})$	$=$	<code>tr(name) tr(ann)</code>
$tr(\langle \text{name, ann, } \{ \text{logExp}_1, \dots, \text{logExp}_m \} \rangle_{\text{axiom}})$	$=$	<code>tr(name) tr(ann)</code> <code>definedBy</code> <code>tr(logExp₁) .</code> <code>...</code> <code>tr(logExp_m) .</code>
$tr(\langle \text{name, ann, ontID, medID, nfp, capability, } \{ \text{interface}_1, \dots, \text{interface}_n \} \rangle_{\text{ws}})$	$=$	<code>webService tr(name) tr(ann) tr(nfp) tr_{io}(ontID) tr_{um}(medID)</code> <code>tr(capability) tr(interface₁) ... tr(interface_n)</code>
$tr(\langle \text{name, ann, ontID, medID, nfp, capability, } \{ \text{interface}_1, \dots, \text{interface}_n \} \rangle_{\text{goal}})$	$=$	<code>goal tr(name) tr(ann) tr(nfp) tr_{io}(ontID) tr_{um}(medID)</code> <code>tr(capability) tr(interface₁) ... tr(interface_n)</code>

Table 2.3: Mapping conceptual syntax to the surface syntax (II)



$tr(\langle \text{name, ann, ontID, medID, nfp, \{sharedVar_1, \dots, sharedVar_n\}, \{pre_1, \dots, pre_m\}, \{post_1, \dots, post_l\}, \{ass_1, \dots, ass_k\}, \{eff_1, \dots, eff_o\} \rangle_{\text{capability}})$	$\text{capability } tr(\text{name}) \text{ } tr(\text{ann}) \text{ } tr(\text{nfp}) \text{ } tr_{io}(\text{ontID}) \text{ } tr_{um}(\text{medID})$ $\text{sharedVariables } \{tr(\text{sharedVar}_1), \dots, tr(\text{sharedVar}_n)\}$ $\text{preCondition } tr(\text{pre}_1) \dots \text{preCondition } tr(\text{pre}_m)$ $\text{postCondition } tr(\text{post}_1) \dots \text{postCondition } tr(\text{post}_l)$ $\text{assumption } tr(\text{ass}_1) \dots \text{assumption } tr(\text{ass}_k)$ $\text{effect } tr(\text{eff}_1) \dots \text{effect } tr(\text{eff}_o)$
$tr(\langle \text{name, ann, ontID, medID, nfp, \{chor_1, \dots, chor_n\}, \{orchID_1, \dots, orchID_m\} \rangle_{\text{interface}})$	$\text{interface } tr(\text{name}) \text{ } tr(\text{ann}) \text{ } tr(\text{nfp}) \text{ } tr_{io}(\text{ontID}) \text{ } tr_{um}(\text{medID})$ $tr(\text{chor}_1) \dots tr(\text{chor}_n)$ $\text{orchestration } tr(\text{orch}_1) \dots \text{orchestration } tr(\text{orch}_m)$
$tr(\langle \text{name, ann, ontID, medID, signature, \{rule_1, \dots, rule_n\} \rangle_{\text{chor}})$	$\text{choreography } tr(\text{name}) \text{ } tr(\text{ann}) \text{ } tr(\text{nfp}) \text{ } tr_{io}(\text{ontID})$ $tr_{um}(\text{medID}) \text{ } tr(\text{signature})$ $\text{transitionRules } tr(\text{rule}_1) \dots tr(\text{rule}_n)$
$tr(\langle \text{name, ann, ontID, medID, \{mode_1, \dots, mode_n\} \rangle_{\text{ss}})$	$\text{stateSignature } tr(\text{name}) \text{ } tr(\text{ann}) \text{ } tr(\text{nfp}) \text{ } tr_{io}(\text{ontID})$ $tr_{um}(\text{medID}) \text{ } tr(\text{mode}_1) \dots tr(\text{mode}_n)$
$tr(\langle \text{type, conceptID, \{groundingID_1, \dots, groundingID_n\} \rangle_{\text{cm}})$	$\text{type concept } tr(\text{conceptID}) \text{ withGrounding}$ $\{tr(\text{groundingID}_1), \dots, tr(\text{groundingID}_n)\}$
$tr(\langle \text{type, relationID, \{groundingID_1, \dots, groundingID_n\} \rangle_{\text{rm}})$	$\text{type relation } tr(\text{relationID}) \text{ withGrounding}$ $\{tr(\text{groundingID}_1), \dots, tr(\text{groundingID}_n)\}$
$tr(\langle \text{logExp, \{rule_1, \dots, rule_n\} \rangle_{\text{if}})$	$\text{if } tr(\text{logExp}) \text{ then } tr(\text{rule}_1) \dots tr(\text{rule}_n) \text{ endif}$
$tr(\langle \{\text{varID}_1, \dots, \text{varID}_n\}, \text{logExp, \{rule_1, \dots, rule_m\} \rangle_{\text{forall}})$	$\text{forall } tr(\text{varID}_1), \dots, tr(\text{varID}_n) \text{ with } tr(\text{logExp}) \text{ do}$ $tr(\text{rule}_1) \dots tr(\text{rule}_m) \text{ endforall}$
$tr(\langle \{\text{varID}_1, \dots, \text{varID}_n\}, \text{logExp, \{rule_1, \dots, rule_m\} \rangle_{\text{choose}})$	$\text{choose } tr(\text{varID}_1), \dots, tr(\text{varID}_n) \text{ with } tr(\text{logExp}) \text{ do}$ $tr(\text{rule}_1) \dots tr(\text{rule}_m) \text{ endChoose}$
$tr(\langle \{\text{rule}_1, \dots, \text{rule}_m \rangle_{\text{pipel}})$	$tr(\text{rule}_1) \dots tr(\text{rule}_m)$
$tr(\langle \text{fact} \rangle_{\text{add}})$	$\text{add}(tr(\text{fact}))$
$tr(\langle \text{fact} \rangle_{\text{delete}})$	$\text{delete}(tr(\text{fact}))$

Table 2.4: Mapping conceptual syntax to the surface syntax (III)

$tr(\langle \text{name, ann, ontID, nfp, \{sourceID_1, \dots, sourceID_n\}, \{targetID_1, \dots, targetID_m\}, \{serviceID_1, \dots, serviceID_l\} \rangle_{\text{oom}})$	$\text{ooMediator } tr(\text{name}) \text{ } tr(\text{ann}) \text{ } tr(\text{nfp}) \text{ } tr_{io}(\text{ontID})$ $tr_{um}(\text{medID}) \text{ source } \{tr(\text{sourceID}_1), \dots, tr(\text{sourceID}_n)\}$ $\text{target } \{tr(\text{targetID}_1), \dots, tr(\text{targetID}_m)\}$ $\text{usesService } \{tr(\text{serviceID}_1), \dots, tr(\text{serviceID}_n)\}$
$tr(\langle \text{name, ann, ontID, nfp, \{sourceID_1, \dots, sourceID_n\}, \{targetID_1, \dots, targetID_m\}, \{serviceID_1, \dots, serviceID_l\} \rangle_{\text{ggm}})$	$\text{ggMediator } tr(\text{name}) \text{ } tr(\text{ann}) \text{ } tr(\text{nfp}) \text{ } tr_{io}(\text{ontID})$ $tr_{um}(\text{medID}) \text{ source } \{tr(\text{sourceID}_1), \dots, tr(\text{sourceID}_n)\}$ $\text{target } \{tr(\text{targetID}_1), \dots, tr(\text{targetID}_m)\}$ $\text{usesService } \{tr(\text{serviceID}_1), \dots, tr(\text{serviceID}_n)\}$
$tr(\langle \text{name, ann, ontID, nfp, \{sourceID_1, \dots, sourceID_n\}, \{targetID_1, \dots, targetID_m\}, \{serviceID_1, \dots, serviceID_l\} \rangle_{\text{wgm}})$	$\text{wgMediator } tr(\text{name}) \text{ } tr(\text{ann}) \text{ } tr(\text{nfp}) \text{ } tr_{io}(\text{ontID})$ $tr_{um}(\text{medID}) \text{ source } \{tr(\text{sourceID}_1), \dots, tr(\text{sourceID}_n)\}$ $\text{target } \{tr(\text{targetID}_1), \dots, tr(\text{targetID}_m)\}$ $\text{usesService } \{tr(\text{serviceID}_1), \dots, tr(\text{serviceID}_n)\}$
$tr(\langle \text{name, ann, ontID, nfp, \{sourceID_1, \dots, sourceID_n\}, \{targetID_1, \dots, targetID_m\}, \{serviceID_1, \dots, serviceID_l\} \rangle_{\text{wmm}})$	$\text{wwMediator } tr(\text{name}) \text{ } tr(\text{ann}) \text{ } tr(\text{nfp}) \text{ } tr_{io}(\text{ontID})$ $tr_{um}(\text{medID}) \text{ source } \{tr(\text{sourceID}_1), \dots, tr(\text{sourceID}_n)\}$ $\text{target } \{tr(\text{targetID}_1), \dots, tr(\text{targetID}_m)\}$ $\text{usesService } \{tr(\text{serviceID}_1), \dots, tr(\text{serviceID}_n)\}$

Table 2.5: Mapping conceptual syntax to the surface syntax (IV)



- If the set of WSML annotations is empty, the keywords `annotation` and `endAnnotation` (resp., `nonFunctionalProperties` and `endNonFunctionalProperties`) may be omitted.
- The keyword `nfp` may be used as shortcut for `nonFunctionalProperty`.
- Singleton sets of identifiers of the form `{id}`, where `id` is an identifier, may be written as `id`.
- Empty ontology import, mediator usage, super concept, super relation, shared variable, grounding, mediator source, mediator target, mediator service usage statements of the forms `importsOntology {}`, `usesMediator {}`, `subConceptOf {}`, `subRelationOf {}`, `sharedVariables {}`, `withGrounding {}`, `source {}`, `target {}`, `usesMediator {}` may be omitted.
- Cardinality statements of the form `(0 *)` may be omitted.
- Cardinality statements of the form `(i *)`, with `i` a nonnegative integer, may be written as `(i)`.
- A concept mode `concept uri`, with `uri` a IRI or `sQName`, may be written as `uri`.
- A set of modes with the same type can be aggregated into one mode, e.g. in `mode1 ... in moden` can be abbreviated to `in mode1, ..., moden`.
- Sets of add or delete rules with molecules about the same identifier can be aggregated into one rule.
- If a Web service description does not have an identifier and does not have any annotations, imported ontologies, used mediators, nonfunctional properties, or interfaces (i.e. it only has a capability), then the keyword `webService` may be discarded. This corresponds to the case of a standalone capability. Note that such a capability must be written in a separate file in order to avoid ambiguities in the syntax.
- If a Web service description does not have an identifier, does not have any annotations, imported ontologies, used mediators, or nonfunctional properties, its capability does not have a identifier and is further empty, and it has exactly one interface, then the keyword `webService` may be discarded. This corresponds to the case of a standalone interface. Note that such an interface must be written in a separate file in order to avoid ambiguities in the syntax.
- If a state signature does not have the name and is otherwise empty, it may be discarded, i.e. the keyword `stateSignature` may be omitted.
- Imported ontologies in capabilities, interfaces, choreographies, and signatures may be discarded if they are imported in the containing Web services/goals, interfaces, or choreographies, respectively.



Abstract Syntax	Surface Syntax
$tr(p(t_1, \dots, t_n))$	$p(t_1, \dots, t_n)$
$tr(\top)$	true
$tr(\perp)$	false
$tr(t_1 = t_2)$	$t_1 = t_2$
$tr(t_1 : t_2)$	t_1 memberOf t_2
$tr(t_1 :: t_2)$	t_1 subConceptOf t_2
$tr(t_1[t_2 \text{ hv } t_3])$	$t_1[t_2$ hasValue $t_3]$
$tr(t_1[t_2 \text{ ot } t_3])$	$t_1[t_2$ ofType $t_3]$
$tr(t_1[t_2 \text{ it } t_3])$	$t_1[t_2$ impliesType $t_3]$
$tr(\neg\phi)$	neg $tr(\phi)$
$tr(\text{not } \phi)$	naf $tr(\phi)$
$tr_{rule}((\forall)b_1 \wedge \dots \wedge b_l \wedge$ $\text{not } c_1 \wedge \dots \wedge \text{not } c_m \supset \perp),$ with $l + m \geq 1$!- $tr(b_1)$ and \dots and $tr(b_l)$ and naf $tr(c_1)$ and \dots and naf $tr(c_m)$
$tr_{rule}((\forall)b_1 \wedge \dots \wedge b_l \wedge$ $\text{not } c_1 \wedge \dots \wedge \text{not } c_m \supset h),$ with $h \neq \perp$	$tr(h)$:- $tr(b_1)$ and \dots and $tr(b_l)$ and naf $tr(c_1)$ and \dots and naf $tr(c_m)$
$tr(\phi \wedge \psi)$	$tr(\phi)$ and $tr(\psi)$
$tr(\phi \vee \psi)$	$tr(\phi)$ or $tr(\psi)$
$tr(\phi \supset \psi)$	$tr(\phi)$ implies $tr(\psi)$
$tr(\phi \equiv \psi)$	$tr(\phi)$ equivalent $tr(\psi)$
$tr(\forall x(\phi))$	forall $?x(tr(\phi))$
$tr(\exists x(\phi))$	exists $?x(tr(\phi))$

Table 2.6: Mapping logical expressions to the surface syntax

2.3 Logical Expression Syntax

Table 2.6 shows the mapping of logical expressions to the surface syntax. In the table, t_1, \dots, t_n are terms, p is a predicate symbol, x is a variable, $b_1, \dots, b_l, c_1, \dots, c_m, h$ are atoms or molecules, and ϕ, ψ are formulas. The translation function tr_{rule} may only (and must) be used if the Flight, Rule, or Full variant of WSMML is considered.

The following *shortcut syntax* may be used in the surface syntax:

- A formula ϕ **implies** ψ may be written as ϕ **impliedBy** ψ .
- Complex formulas in both the bodies and the heads of the rules of WSMML-Flight and WSMML-Rule ontologies. These can be eliminated through the usual Lloyd-Topor transformations described in the specification.
- Conjunctions of attributes molecules of the same type about the same term and the same attribute $t[s \times r_1]$ and \dots **and** $t[s \times r_n]$ may be combined: $t[s \times \{r_1, \dots, r_n\}]$, where x is either **hasValue**, **impliesType** or **ofType**.
- Conjunctions of attributes molecules about the same term $t[s_1 \times_1 r_1]$ and \dots **and** $t[s_n \times_n r_n]$ may be combined to create a compound attribute molecule: $t[s_1 \times_1 r_1, \dots, s_n \times_n r_n]$, where x_1, \dots, x_n are **hasValue**, **impliesType** or **ofType**.



- The conjunction of a sub-concept or membership molecule and an attribute molecule about the same term $t[s_1 x_1 r_1, \dots, s_n x_n r_n]$ and $t y u$ may be combined to create a compound molecule: $t[s_1 x_1 r_1, \dots, s_n x_n r_n] y u$, where y is either `memberOf` or `subConceptOf`.
- A rule with an empty body $A :- .$ may be written as A .
- Multiple rules with the same body $a_1 :- B \dots a_n :- B$ may be written as one rule $a_1 \text{ and } \dots \text{ and } a_n :- B$.
- A rule of the form $A :- B$ and a may be written as $a \text{ implies } A :- B$.
- Two rules of the forms $A_1 \text{ implies } A_2 :- B$ and $A_2 \text{ implies } A_1 :- B$ may be written as one rule $A_1 \text{ equivalent } A_2 :- B$.

If the considered variant is `Flight`, `Rule`, or `Full`, then the following shortcut syntax may additionally be used:

- Two rules of the forms $A :- B_1$ and $B' \text{ and } B_3$ and $A :- B_1$ and $B'' \text{ and } B_3$ may be written as one rule $A :- B_1$ and $(B' \text{ or } B'') \text{ and } B_3$.
- Two rules of the forms $A :- B_1$ and $\text{naf } B' \text{ and } B_3$ and $A :- B_1$ and $\text{naf } B'' \text{ and } B_3$ may be written as one rule $A :- B_1$ and $\text{naf } (B' \text{ and } B'') \text{ and } B_3$.
- A rule of the form $A :- B_1$ and $\text{naf } B' \text{ and } \text{naf } B'' \text{ and } B_3$ may be written as $A :- B_1$ and $\text{naf } (B' \text{ or } B'') \text{ and } B_3$.
- A rule of the form $A :- B_1$ and $B' \text{ and } B_3$ may be written as $A :- B_1$ and $\text{naf } \text{naf } B' \text{ and } B_3$.

If the considered variant is `Rule` or `Full`, then the following shortcut syntax may additionally be used:

- Two rules of the forms $A :- B_1$ and $\text{naf } B' \text{ and } B_3$ and $A :- B_1$ and $B'' \text{ and } B_3$ may be written as one rule $A :- B_1$ and $(B' \text{ implies } B'') \text{ and } B_3$.
- Two rules of the forms $A :- B_1$ and $\text{naf } B' \text{ and } B_3$ and $A :- B_1$ and $B'' \text{ and } B_3$ may be written as one rule $A :- B_1$ and $(B' \text{ impliedBy } B'') \text{ and } B_3$.
- Two rules of the forms $A :- B_1$ and $(B' \text{ implies } B'') \text{ and } B_3$ and $A :- B_1$ and $(B' \text{ impliedBy } B'') \text{ and } B_3$ may be written as one rule $A :- B_1$ and $(B' \text{ equivalent } B'') \text{ and } B_3$.
- A rule of the form $A :- B_1$ and $B' \text{ and } B_3$ may be written as rule $A :- B_1$ and $\text{exists } ?x_1, \dots, ?x_n (B') \text{ and } B_3$.
- A rule of the form $A :- B_1$ and $\text{naf } B' \text{ and } B'' \text{ and } B_3$ may be written as $A :- B_1$ and $\text{naf } B'' \text{ implies } B' \text{ and } B_3$.
- Two rules of the forms $A :- B_1$ and $\text{naf } p(?x_1, \dots, ?x_n) \text{ and } B_3$ and $p(?y_1, \dots, ?y_n) :- \text{exists } ?x_1, \dots, ?x_m (B')$ such that p does not occur anywhere else in the ontology or any of the imported ontologies and $?y_1, \dots, ?y_n$ are the free variables in $\text{exists } ?x_1, \dots, ?x_m (B')$, may be written as one rule $A :- B_1$ and $\text{naf exists } ?x_1, \dots, ?x_m (B') \text{ and } B_3$.
- A rule of the form $A :- B_1$ and $\text{naf exists } ?x_1, \dots, ?x_m (\text{naf } B') \text{ and } B_3$ may be written as $A :- B_1$ and $\text{forall } ?x_1, \dots, ?x_n (B') \text{ and } B_3$.



3 WSML Semantics

In this chapter we define the semantics of ontologies, nonfunctional properties, and choreographies.

- The semantics of WSML ontologies is defined using a model theory. This model theory is combined with the model theories of RDF and OWL DL to obtain an overall model theory for sets of ontologies, which may include WSML, RDFS, and OWL DL ontologies. Notions of satisfiability and entailment are defined for each of the WSML variants. These notions are used for defining the semantics of nonfunctional properties and choreographies. Thereby, the definition of their semantics are independent from the specific variant which has been chosen.
- The semantics of nonfunctional properties is defined in terms of values which satisfy certain conditions. In effect, a nonfunctional property is seen as a query over a set of ontologies and data sources. The definition of the nonfunctional property semantics is orthogonal to that of the ontology semantics; it depends on an entailment relation.
- The semantics of choreographies is defined in terms of interface runs, which are possible interactions between service requesters and providers. As is the case for the nonfunctional property semantics, the semantics for choreographies is parametric with respect to the entailment relation given by the ontology semantics.

The layering of the WSML language variants is discussed in Appendix B.

3.1 Ontology Semantics

In this section we define the semantics of WSML ontologies. This semantics is combined with the RDFS and OWL DL semantics in the following section.

The semantics of WSML ontologies is based on a model theory, similar to the standard model theory used for classical first-order predicate logic (see, e.g., [9]), extended with certain features inspired by the model theory of F-Logic [15].

Recall that different WSML variants are based on different knowledge representation paradigms. It turns out that, because of the differences between the paradigms, it is not known how to use the same model theory for all variants. However, there is a basic notion of a WSML interpretation and there is a basic notion of a model. Different variants extend these definitions in different ways. WSML-DL imposes certain additional restrictions on interpretations, such as a separation of the interpretation of concept, instance, and attribute identifiers, but uses the basic notion of a model. WSML-Core, Flight, and Rule use the basic notion of WSML interpretation, but impose certain additional restrictions on conditions under which an interpretation is a model; models need to be minimal or *stable*. The latter is necessary for the nonmonotonic interpretation of the default negation symbol *not*.



The model theory of WSML-DL has been constructed in such a way that it is close to the usual model theory of the first-order variant [4] of the Description Logic $\mathcal{SHIQ}(\mathbf{D})$ [14]. See Appendix A for a first-order style definition of the $\mathcal{SHIQ}(\mathbf{D})$ semantics.

WSML-Core, Flight, and Rule share the same model theory, which has been constructed in such a way that there is a correspondence with the Stable Model Semantics of normal logic programs [11], i.e., logic programs with no disjunction in the head and with negation in the body of the rules. Note that WSML-Flight theories are locally stratified, and that for locally stratified logic programs the Stable Model Semantics corresponds with the other prominent semantics for logic programs with negation, such as the Well-Founded Semantics [10] and the perfect model semantics [19]. Finally, WSML-Core theories do not contain negation; for negation-free logic programs the Stable Model Semantics corresponds with the minimal Herbrand model semantics [17].

Even though there is a proposal for a semantics for WSML-Full [6, 5], there is no consensus in the research community as to what is the right semantics for combinations of the Description Logic (WSML-DL) ontologies and rules with nonmonotonic negation (WSML-Rule). Therefore, WSML does not specify a semantics for its Full variant. Section 3.1.5 outlines a number of conditions any reasonable semantics for WSML-Full must fulfill.

We first describe the basic WSML model theory. We proceed with a description of the extension to the WSML-DL model theory and the notion of entailment in WSML-DL. We then describe the notions of minimal Herbrand model and stable model for WSML-Core, Flight, and Rule, and define the notion of entailment for these variants. Finally, we extend model theory to account for RDFS and OWL DL ontologies. See Appendix B for a discussion of language layering in WSML.

3.1.1 WSML Variant

Every RDFS ontology has the WSML variant Flight. If an OWL DL ontology is in the DLP subset, i.e. it is equivalent to a set of equality-free Horn formulas, it has the WSML variant Core; otherwise it has the WSML variant DL.

There is a partial order between WSML variants: $Core \prec Flight \prec Rule \prec Full$; $Core \prec DL \prec Full$. If $A \prec B$, then B is a *higher* variant than A .

The WSML variant of a set of ontologies \mathbf{O} is the highest among the variants of the ontologies in \mathbf{O} in case there is a single highest variant; otherwise, the variant of \mathbf{O} is WSML Full.

3.1.2 The Basic WSML Model Theory

The basic WSML model theory defines a notion of interpretation and a notion of satisfaction, which are conditions under which a WSML interpretation is a model of a formula, or a set of formulas.

As usual, an interpretation has an *abstract domain of interpretation*, which is an abstract set of objects used for the interpretation of IRIs. Additionally, an interpretation has a *concrete domain*, which is used for the interpretation of



data values. Different interpretations may have different abstract domains and different mappings of IRIs to this abstract domain. The concrete domains of all interpretations under consideration must be the same, as must be the mapping of data values to elements of this domain. This means that any data value is mapped to the same element (value) in the concrete domain, in every interpretation under consideration. For example, an IRI $ex\#abc$ may be mapped to some abstract object a in one interpretation and to b in some other interpretation, but the integer 1 must be mapped to the number 1 in every interpretation.

The concrete domain to be considered is given by a *concrete domain scheme*, which defines a concrete domain, a mapping of data values to elements of this domain, and a number of built-in functions and predicates (e.g., numeric addition, string concatenation, and numeric comparison). Since the list of datatypes that may be used with WSML is not fixed, there is not a single concrete domain scheme for WSML. However, WSML does impose certain requirements on so-called *WSML-compliant* domain schemes, namely, the scheme must include the XML schema datatypes *string*, *integer*, and *decimal* [3], the RDF datatype *XMLLiteral*, and a number of built-in predicates defined for WSML (see [20, Appendix B.2]).

We now proceed to define the notions of interpretation, satisfaction, and concrete domain scheme.

3.1.2.1 WSML Interpretations

An interpretation is a tuple $\mathcal{I} = \langle U, \prec_U, \in_U, U^D, \mathcal{I}_F, \mathcal{I}_P, \mathcal{I}_{hv}, \mathcal{I}_{it}, \mathcal{I}_{ot} \rangle$, where

- U is a nonempty countable set, called the *abstract* domain,
- U^D is a non-empty set that is disjoint from U , called the *concrete* domain,
- \prec_U is an irreflexive partial order over $U \cup U^D$, representing the strict sub-concept relation,
- \in_U is a binary relation over $U \cup U^D$, representing the concept membership relation,
- \mathcal{I}_F is a mapping from constant and function identifiers to elements of U and functions over $(U \cup U^D)$,
- \mathcal{I}_P is a mapping from relation identifiers to relations over $(U \cup U^D)$, and
- \mathcal{I}_{hv} , \mathcal{I}_{it} , and \mathcal{I}_{ot} are mappings from $U \cup U^D$ to binary relations over $U \cup U^D$, representing the attribute value (*hasValue*) and the two kinds of attributes typing relations (*impliesType* and *ofType*): $\mathcal{I}_{hv}, \mathcal{I}_{it}, \mathcal{I}_{ot} : U \cup U^D \rightarrow 2^{(U \cup U^D) \times (U \cup U^D)}$.

Additionally, the following two conditions ((3.1) and (3.2)) must hold on every WSML interpretation. These conditions that WSML interpretations obey the semantics of the sub-concept and *impliesType* relations. The first condition requires that if the sub-concept relation holds between two elements, the set of instances of the sub-concept is a subset of the set of instances of the super-concept.



We write $a \preceq_U b$ when $a \prec_U b$ or $a = b$, for any two $a, b \in U \cup U^D$. For every interpretation must hold that

$$\text{if } a \in_U b \text{ and } b \preceq_U c \text{ then } a \in_U c \quad (3.1)$$

The second condition requires that if the `impliesType` typing relation holds between an attribute p and a concept d , for a concept c , then it must be the case that for every instance a of c , whenever the attribute p has a value b , then b must be an instance of d :

if $\langle c, d \rangle \in \mathcal{I}_{\text{it}}(p)$, then for every $a \in_U c$ holds that

$$\text{for every } b \in U \cup U^D \text{ such that } \langle a, b \rangle \in \mathcal{I}_{\text{hv}}(p), b \in_U d \quad (3.2)$$

From condition 3.1 follows that, if $b \preceq_U c$, then the set of instances of b is a subset of the set of instances of c , i.e., $\{k \mid k \in_U b, k \in U \cup U^D\} \subseteq \{k \mid k \in_U c, k \in U \cup U^D\}$. We call the set $\{k \mid k \in_U b, k \in U \cup U^D\}$ the *class extension* of b , and denote the class extension of an element b with b_{cext} . Thus,

$$\text{if } b \preceq_U c, \text{ then } b_{\text{cext}} \text{ is a subset of } c_{\text{cext}} \quad (3.3)$$

However, the converse of (3.3) is not always true: if b_{cext} is a subset of c_{cext} , then it is not necessarily the case that $b \preceq_U c$.

Note that a consequence is that it is not possible to derive a sub-concept statement $c :: d$ from a formula $\forall x(x : c \supset x : d)$. Note that this kind of derivation is common in Description Logics; in fact, in DLs there is no distinction between sub-concept statements and formulas of the mentioned form; both are written as $c \sqsubseteq d$. Consequently, WSML-DL requires an additional conditions that ensures the converse of 3.3 is be true in every interpretation; see the following subsection.

Before proceeding with the definitions of the interpretation of identifiers, we must make a note about anonymous identifiers and the symbol *nil*. In the following we assume that

- each occurrence of the symbol *nil* is replaced with a globally unique new IRI,
- each occurrence of an unnumbered anonymous identifier *.#* is replaced with a globally unique new IRI, and
- for every formula ϕ and every numbered anonymous identifier *.#n* occurring in ϕ , each occurrence of *.#n* is replaced with the same new globally unique IRI.

We now proceed with the definition of the interpretation functions for identifiers, namely \mathcal{I}_F and \mathcal{I}_P . Note that all identifiers (after replacement of anonymous identifiers) are IRIs, with the exception of elementary data values.

An instance identifier is interpreted as an element of the abstract domain U : $\mathcal{I}_F(f) = u \in U$. A function identifier is interpreted as a function over the domain U , for every arity $i \geq 1$: $\mathcal{I}_F(f)^i : U^i \rightarrow U$. An n -ary datatype wrapper or elementary data value (elementary data values have arity 0) f is interpreted as a function over the domain U^D : $\mathcal{I}_F(f) : (U^D)^n \rightarrow U^D$. A relation identifier p is interpreted as a relation over the domain $U \cup U^D$ for every arity $i \geq 0$:



$\mathcal{I}_P(p)^i \subseteq (U \cup U^D)^i$. An n -ary built-in predicate identifier p is interpreted as a relation over the domain U^D : $\mathcal{I}_P(p) \subseteq (U^D)^n$.

After defining the interpretation of constant, predicate, and function symbols, we can now define the interpretation of terms. To do that, we first need to define the notion of variable assignment. We need to distinguish between *abstract* and *concrete* assignments.

A variable assignment B assigns each variable x to an individual $x^B \in U \cup U^D$. A variable assignment B' is an abstract (resp., concrete) x -variant of B if $x^{B'} \in U \cup U^D$ (resp., $x^{B'} \in U^D$) and $y^{B'} = y^B$ for every $y \neq x$.

We are now ready to define interpretation of terms.

The interpretation of a term t in some interpretation \mathcal{I} with respect to some variable assignment B , written $t^{\mathcal{I},B}$, is defined as: $t^{\mathcal{I},B} = t^B$ if $t \in \mathcal{V}$, and $t^{\mathcal{I},B} = \mathcal{I}_F(f)(t_1^{\mathcal{I},B}, \dots, t_n^{\mathcal{I},B})$ if t is of the form $f(t_1, \dots, t_n)$, with $n \geq 0$.

3.1.2.2 Satisfaction

Satisfaction is a relation between interpretations and formulas. Whenever a pair of an interpretation \mathcal{I} and a formula ϕ are in the satisfaction relation, the interpretation \mathcal{I} *satisfies* the formula ϕ , denoted $\mathcal{I} \models \phi$. If $\mathcal{I} \models \phi$ we say that \mathcal{I} is a *model* of ϕ ; in other words, ϕ is *true* in \mathcal{I} . If \mathcal{I} and ϕ are not in the satisfaction relation, i.e., \mathcal{I} is not a model of ϕ , we write $\mathcal{I} \not\models \phi$.

To define satisfaction of formulas with free variables, we formally define the satisfaction relation relative to a given variable assignment B . We first define satisfaction of atomic formulas and molecules and subsequently extend it to arbitrary formulas.

Satisfaction of atomic formulas and molecules ϕ in an interpretation \mathcal{I} , given a variable assignment B , denoted $(\mathcal{I}, B) \models \phi$, is defined as:

- $(\mathcal{I}, B) \models \top$,
- $(\mathcal{I}, B) \not\models \perp$,
- $(\mathcal{I}, B) \models p(t_1, \dots, t_n)$ iff $(t_1^{\mathcal{I},B}, \dots, t_n^{\mathcal{I},B}) \in \mathcal{I}_P(p)$,
- $(\mathcal{I}, B) \models t_1 : t_2$ iff $t_1^{\mathcal{I},B} \in_U t_2^{\mathcal{I},B}$,
- $(\mathcal{I}, B) \models t_1 :: t_2$ iff $t_1^{\mathcal{I},B} \preceq_U t_2^{\mathcal{I},B}$,
- $(\mathcal{I}, B) \models t_1 [t_2 \text{ hv } t_3]$ iff $\langle t_1^{\mathcal{I},B}, t_3^{\mathcal{I},B} \rangle \in \mathcal{I}_{\text{hv}}(t_2^{\mathcal{I},B})$,
- $(\mathcal{I}, B) \models t_1 [t_2 \text{ it } t_3]$ iff $\langle t_1^{\mathcal{I},B}, t_3^{\mathcal{I},B} \rangle \in \mathcal{I}_{\text{it}}(t_2^{\mathcal{I},B})$,
- $(\mathcal{I}, B) \models t_1 [t_2 \text{ ot } t_3]$ iff $\langle t_1^{\mathcal{I},B}, t_3^{\mathcal{I},B} \rangle \in \mathcal{I}_{\text{ot}}(t_2^{\mathcal{I},B})$, and
- $(\mathcal{I}, B) \models t_1 = t_2$ iff $t_1^{\mathcal{I},B} = t_2^{\mathcal{I},B}$.

This extends to arbitrary formulas as follows:

- $(\mathcal{I}, B) \models \phi_1 \wedge \phi_2$ iff $(\mathcal{I}, B) \models \phi_1$ and $(\mathcal{I}, B) \models \phi_2$,
- $(\mathcal{I}, B) \models \phi_1 \vee \phi_2$ iff $(\mathcal{I}, B) \models \phi_1$ or $(\mathcal{I}, B) \models \phi_2$,
- $(\mathcal{I}, B) \models \phi_1 \supset \phi_2$ iff $(\mathcal{I}, B) \not\models \phi_1$ or $(\mathcal{I}, B) \models \phi_2$,



- $(\mathcal{I}, B) \models \phi_1 \equiv \phi_2$ iff $(\mathcal{I}, B) \models \phi_1 \supset \phi_2$ and $(\mathcal{I}, B) \models \phi_2 \supset \phi_1$,
- $(\mathcal{I}, B) \models \neg\phi_1$ iff $(\mathcal{I}, B) \not\models \phi_1$,
- $(\mathcal{I}, B) \models \text{not } \phi_1$ iff $(\mathcal{I}, B) \not\models \phi_1$,
- $(\mathcal{I}, B) \models \forall_a x(\phi_1)$ iff for every B'_a , which is an abstract x -variant of B , $(\mathcal{I}, B'_a) \models \phi_1$,
- $(\mathcal{I}, B) \models \exists_a x(\phi_1)$ iff for some B'_a , which is an abstract x -variant of B , $(\mathcal{I}, B'_a) \models \phi_1$,
- $(\mathcal{I}, B) \models \forall_c x(\phi_1)$ iff for every B'_c , which is a concrete x -variant of B , $(\mathcal{I}, B'_c) \models \phi_1$,
- $(\mathcal{I}, B) \models \exists_c x(\phi_1)$ iff for some B'_c , which is a concrete x -variant of B , $(\mathcal{I}, B'_c) \models \phi_1$.

If a variable x is quantified using a concrete quantifier (\forall_c, \exists_c), we call x a *concrete variable*; otherwise, we call x an *abstract variable*.

An interpretation \mathcal{I} satisfies a formula ϕ , written $\mathcal{I} \models \phi$ if $(\mathcal{I}, B) \models \phi$ for every variable assignment B .

Notice that we did not say anything about the concrete domain or the interpretation of the concrete domain functions and predicates. We now proceed to define the notion of concrete domain schemes and the notion of models relative to concrete domain schemes.

3.1.2.3 Concrete Domain Schemes

A concrete domain scheme consists of a concrete domain, which is a set of values (e.g., integers and strings), a set of concrete predicate symbols (e.g., numeric comparison such as greater-than), a set of data values identifiers (e.g., strings and integers), a set of datatype identifiers, a set of concrete function symbols, and an interpretation function that maps concrete function symbols to functions and concrete predicate symbols to relations.

We note that, generally, WSML uses concrete predicate symbols for built-in functions. For example, numeric addition $x + y = z$ correspond to an atomic formula with a concrete predicate symbol `wsml#numericAdd(z, x, y)`. Concrete function symbols are used as constructors of data values (they are *datatype wrappers*).

Formally, a *concrete domain scheme* \mathfrak{S} is a tuple $\mathfrak{S} = \langle U^\mathfrak{S}, \mathcal{F}^\mathfrak{S}, \mathcal{D}^\mathfrak{S}, \mathcal{P}^\mathfrak{S}, \mathcal{D}^\mathfrak{S}, \cdot^\mathfrak{S} \rangle$, where

- $U^\mathfrak{S}$ is a non-empty set of concrete values,
- $\mathcal{F}^\mathfrak{S}$ and $\mathcal{P}^\mathfrak{S}$ are disjoint sets of concrete function and predicate symbols, which are IRIs, each with an associated nonnegative arity n ,
- $\mathcal{D}^\mathfrak{S} \subseteq \mathcal{F}^\mathfrak{S}$ is a set of datatype IRIs, and
- $\cdot^\mathfrak{S}$ is an interpretation function which assigns a function $f^\mathfrak{S} : (U^\mathfrak{S})^n \rightarrow U^\mathfrak{S}$ to every $f \in \mathcal{F}^\mathfrak{S}$, and a relation $p^\mathfrak{S} \subseteq (U^\mathfrak{S})^n$ to every $p \in \mathcal{P}^\mathfrak{S}$.



As an abuse of notation, for every datatype IRI $d \in \mathcal{D}^{\mathfrak{S}}$, with $d^{\mathfrak{S}}$ we denote both the function assigned to d by $\cdot^{\mathfrak{S}}$ and the range of this function. In the latter case, we also speak about $d^{\mathfrak{S}}$ as the *domain* of the datatype identified by d . In other words, we use the same identifier d to denote both the datatype and the datatype wrapper.

Intuitively, a datatype function $d^{\mathfrak{S}}$ defines a datatype as a set of values that are *constructed* from other values. For example, a date 2008-02-13 is constructed from the integers 2008, 2, and 13 using the function `xsd#date`: `xsd#date(2008, 2, 13)`. In addition, `xsd#date` denote the range of the constructor function `xsd#date`, which is the set of all dates, i.e., the value space of the datatype `xsd#date`.

We illustrate the concept of concrete domain schemes through the definition of a scheme for integers and strings.

Example 1. We define $\mathfrak{S} = \langle U^{\mathfrak{S}}, \mathcal{F}^{\mathfrak{S}}, \mathcal{D}^{\mathfrak{S}}, \mathcal{P}^{\mathfrak{S}}, \mathcal{D}^{\mathfrak{S}}, \cdot^{\mathfrak{S}} \rangle$ as follows: $U^{\mathfrak{S}}$ is the union of the sets of integer numbers and finite-length sequences of Unicode characters. $\mathcal{F}^{\mathfrak{S}}$ is the union of the set of finite-length sequences of decimal digits, optionally with a leading minus (-), and the set of finite-length sequences of Unicode characters, delimited with " (for simplicity, we assume that the character " does not occur in such strings), all with arity 0. $\mathcal{P}^{\mathfrak{S}}$ consists of unary predicate symbols `xsd#integer` and `xsd#string`, and the binary predicate symbol `wsml#numeric-equals`. The interpretation function $\cdot^{\mathfrak{S}}$ interprets (signed) sequences of decimal digits and "-delimited sequences of characters as integers and strings, respectively, in the natural way; $\cdot^{\mathfrak{S}}$ interprets the unary predicate symbols `xsd#integer` and `xsd#string` as the sets of integers and strings, respectively; finally, $\cdot^{\mathfrak{S}}$ interprets `wsml#numeric-equals` as identity over the set of integers.

WSML Compliance It is not allowed to use just any kind of concrete domain scheme with WSML. The scheme must at least include the datatypes and built-in predicates required by WSML. Additionally, if any XML schema datatype is used, the scheme must conform to the definition of the datatype in the XML schema datatypes specification [3].

Formally, a concrete domain scheme $\mathfrak{S} = \langle U^{\mathfrak{S}}, \mathcal{F}^{\mathfrak{S}}, \mathcal{D}^{\mathfrak{S}}, \mathcal{P}^{\mathfrak{S}}, \mathcal{D}^{\mathfrak{S}}, \cdot^{\mathfrak{S}} \rangle$ is *WSML-compliant* if the following conditions are met. We first define the condition which make sure all XML schema datatypes in \mathfrak{S} conform with the specification. The IRI of an XML schema datatype is obtained by concatenating XML schema namespace and the name of the datatype. For example, the IRI of the datatype `string` is `http://www.w3.org/2001/XMLSchema#string`.

- if $d \in \mathcal{D}^{\mathfrak{S}}$ is the IRI of an XML schema datatype dt , then the range of $d^{\mathfrak{S}}$ corresponds to the value space of dt and the domain of $d^{\mathfrak{S}}$ corresponds to the definition in [20, Table B.1], if it exists; otherwise it is the set of strings comprising the lexical space of dt , and the mapping $d^{\mathfrak{S}}$ corresponds to the lexical-to-value mapping for dt defined in [3].

The following conditions ensure that the required data types, namely the XML schema datatypes `string`, `integer`, and `decimal` and the RDF datatype `XMLLiteral`, are included in the scheme.

- the IRIs `xsd#string`, `xsd#integer`, and `xsd#decimal` are included in $\mathcal{D}^{\mathfrak{S}}$, and for any string, integer, or decimal v in $\mathcal{F}^{\mathfrak{S}}$, $v^{\mathfrak{S}}$ is the value obtained



by applying the corresponding lexical to value mapping, as defined in [3], to v ,

- the IRI `rdf#XMLLiteral` is included in $\mathcal{D}^{\mathfrak{S}}$, every string x representing valid XML content [16, Section 1] is included in $\mathcal{F}^{\mathfrak{S}}$ with arity 0, and for any string x representing valid XML content, `rdf#XMLLiteral` ^{\mathfrak{S}} (x) is the XML value of x according to [16, Section 5.1], and
- the built-ins defined for WSML in Appendix C.2 of [20] are included in $\mathcal{P}^{\mathfrak{S}}$ and are interpreted to the definition in Appendix C.2 of [20].

In the remainder, we require that every domain scheme under consideration is WSML-compliant.

Conformance of Interpretations The intention of concrete values and concrete predicate and function symbols is that they are interpreted in the same way in all interpretations under consideration. The way this is achieved in WSML is by associating a single concrete domain scheme with all considered interpretations. A concrete domain scheme is associated with an interpretation if the interpretation *conforms* with the scheme, is defined below.

An interpretation $\mathcal{I} = \langle U, \prec_U, \in_U, U^D, \mathcal{I}_F, \mathcal{I}_P, \mathcal{I}_{hv}, \mathcal{I}_{it}, \mathcal{I}_{ot} \rangle$ *conforms to* a concrete domain scheme \mathfrak{S} if the following conditions are satisfied.

- $U^D = U^{\mathfrak{S}}$,
- $\mathcal{I}_F(f) = f^{\mathfrak{S}}$ for every concrete n -ary function symbol $f \in \mathcal{F}^{\mathfrak{S}}$, and
- $\mathcal{I}_P(p) = p^{\mathfrak{S}}$ for every concrete n -ary predicate symbol $p \in \mathcal{P}^{\mathfrak{S}}$.

We note here that a datatype identifier is also a concept identifier, and the set of instances of this concept is the set of values of the datatype. This is guaranteed by the following condition.

- for every datatype identifier $d \in \mathcal{D}^{\mathfrak{S}}$ holds that $u \in d^{\mathfrak{S}}$ iff $u \in_U \mathcal{I}_F(d)$, for every $u \in U^D$.

3.1.2.4 Models

Using the above notion of conformance we can now define the notion of a model relative to a concrete domain scheme.

Given a concrete domain scheme \mathfrak{S} , an interpretation \mathcal{I} is a *\mathfrak{S} -model* of a formula ϕ if \mathcal{I} conforms to \mathfrak{S} and $\mathcal{I} \models \phi$. A formula ϕ is *\mathfrak{S} -satisfiable* if it has a \mathfrak{S} -model; ϕ is *\mathfrak{S} -valid* if every interpretation that conforms to \mathfrak{S} is a \mathfrak{S} -model of ϕ .

Likewise, an interpretation \mathcal{I} is a \mathfrak{S} -model of a theory Φ if \mathcal{I} is a \mathfrak{S} -model of every formula $\phi \in \Phi$ and Φ is \mathfrak{S} -satisfiable if it has a \mathfrak{S} -model.

We have defined the basic notion of models in WSML. The subsequent sections extend this notion in two different ways, for the DL and Core, Flight, and Rules variants, respectively. These extended notions will be used to define the semantic notions for the respective variants.



3.1.3 WSML-DL Extensions

WSML-DL makes a syntactic separation between concepts, attributes, and instance identifiers, and the places they may occur (e.g., a concept identifier may only occur in a concept positions). Additionally, the places where abstract and concrete terms and variables may occur are distinct. In order to enable DL-based reasoning it is necessary to reflect these distinctions in the semantics as well. Specifically, WSML-DL imposes additional conditions on interpretations.

We first need to define the notions of concept, attribute, and instance position. We say that a term occurs in a *concept position* if it occurs as the third term in a `it` or `ot` molecule, as the second term in a `:-` molecule, or the term occurs in a `::` molecule; it occurs in an *attribute position* if it occurs as the second term in a `hv`, `it` or `ot` molecule; otherwise, the term occurs in an *instance position*. That is, in the molecules and atoms $a:c$, $c::c$, $a[\text{phv}a]$, $a[\text{pit}c]$, $a[\text{pot}c]$, $q(a, \dots, a)$, and $a = a$ the term c occurs in concept, p occurs in relation, and a occurs in instance positions.

3.1.3.1 WSML-DL Semantic conditions

Observe that the syntax of WSML-DL makes a distinction between identifiers used in concept, attribute and instance positions. In fact, the sets of instances, concept, attribute, and annotation property identifiers are mutually disjoint. WSML interpretations, defined in the previous subsection, do not distinguish between the interpretations of these different identifiers; they are all interpreted in the same domain. This separation is reflected in the conditions on WSML-DL interpretations, as well as conditions on WSML-DL variable assignments.

An interpretation $\mathcal{I} = \langle U, \prec_U, \in_U, U^D, \mathcal{I}_F, \mathcal{I}_P, \mathcal{I}_{\text{hv}}, \mathcal{I}_{\text{it}}, \mathcal{I}_{\text{ot}} \rangle$ is a *WSML-DL interpretation* if the following conditions hold. Firstly, we require a separation between the interpretations of instance, concepts, and attributes identifiers.

- U is partitioned into three sub-domains U^i , U^a , and U^c , such that U^i is not empty,
- \mathcal{I}_F maps instance identifiers to elements in U^i ,
- \mathcal{I}_F maps concept identifiers to elements in U^c , and
- \mathcal{I}_F maps attribute and annotation property identifiers to elements in U^a .

Then, we need to ensure that the elements of the interpretation that are used for the interpretation of subclass, class instance, and attribute statements are defined only on the appropriate sub-domains.

- the partial order \prec_U is only defined on elements in U^c ,
- \in_U is a relation between U^i and U^c , and
- $\mathcal{I}_{\text{hv}}(u) = \mathcal{I}_{\text{it}}(u) = \mathcal{I}_{\text{ot}}(u) = \emptyset$ for any $u \in U^c \cup U^i$.

Finally, WSML-DL requires an *extensional* interpretation of the subclass and implies-type constructs, i.e., the converse of the conditions (3.1) and (3.2) must



hold:

for any two elements $a, b \in U^c$ holds that

$$\text{whenever } a_{\text{cext}} \subseteq b_{\text{cext}}, a \preceq_U b \quad (3.4)$$

for any three elements $c, p, d \in U$ it is the case that

$$\begin{aligned} &\text{whenever for every } a, b \in U \text{ such that } a \in_U c \text{ and } \langle a, b \rangle \in \mathcal{I}_{\text{hv}}(p) \\ &\text{it holds that } b \in_U d, \langle c, d \rangle \in \mathcal{I}_{\text{it}}(p) \quad (3.5) \end{aligned}$$

3.1.3.2 WSML-DL Satisfaction

The conditions on WSML-DL interpretations are not yet enough to define satisfaction for WSML-DL. Namely, it is necessary to ensure that variables are only mapped to the sub-domain U^i and the concrete domain U^D , and abstract variables must only be mapped to U^i .

Formally, given a variable assignment B , a variable assignment B' is an abstract DL- x -variant of B if B' is an abstract x -variant of B and $x^{B'} \in U^i$.

DL satisfaction, denoted by the symbol \models_{DL} , is obtained from satisfaction as described in the previous section by modifying the definition of satisfaction of abstractly quantified formulas in the following way:

- $(\mathcal{I}, B) \models_{DL} \forall_a x(\phi_1)$ iff for every variable assignment B'_a , which is an abstract DL- x -variant of B whose range is U^i , $(\mathcal{I}, B'_a) \models_{DL} \phi_1$ and
- $(\mathcal{I}, B) \models_{DL} \exists_a x(\phi_1)$ iff for some variable assignment B'_a , which is an abstract DL- x -variant of B whose range is U^i , $(\mathcal{I}, B'_a) \models \phi_1$.

As can be seen from the definition, the difference with the definition of satisfaction given earlier is the following: abstractly quantified variables are only assigned to the individual domain U^i , and not to the concrete domain U^D , the concept domain U^c , or the attribute domain U^a . The notions of a model, satisfiability, and validity in WSML-DL are defined analogously to the corresponding notions in WSML.

Given a concrete domain scheme \mathfrak{S} , an interpretation \mathcal{I} is a *WSML-DL \mathfrak{S} -model* of a WSML-DL formula ϕ if \mathcal{I} is a WSML-DL interpretation, \mathcal{I} conforms to \mathfrak{S} , and $\mathcal{I} \models_{DL} \phi$. A formula ϕ is *WSML-DL \mathfrak{S} -satisfiable* if it has a WSML-DL \mathfrak{S} -model; ϕ is *WSML-DL \mathfrak{S} -valid* if every WSML-DL interpretation which conforms to \mathfrak{S} is a model of ϕ .

Likewise, an interpretation \mathcal{I} is a \mathfrak{S} -model of a theory Φ if \mathcal{I} is a \mathfrak{S} -model of every formula $\phi \in \Phi$ and Φ is \mathfrak{S} -satisfiable if it has a \mathfrak{S} -model.

3.1.3.3 Semantics of WSML-DL Ontologies

We are now ready to define the semantics of WSML-DL ontologies. The main notions we are interested in are (1) satisfiability of a concept relative to an ontology, (2) satisfiability of an ontology, and (3) entailment between ontologies.

The definitions of these notions follow straightforwardly from the above definitions with the complication that we need to take imported ontologies into account. We first define the notion of a model of a WSML-DL ontology.



Given a concrete domain scheme \mathfrak{S} , an interpretation \mathcal{I} is a *WSML-DL \mathfrak{S} -model* of a WSML-DL ontology O if

1. \mathcal{I} is a WSML-DL \mathfrak{S} -model of a theory Φ which corresponds to O and
2. \mathcal{I} is a WSML-DL \mathfrak{S} -model of every ontology imported by O .

Notice that we require every model of an ontology to be also a model of every imported ontology. Consequently, the interpretation must also be a model of every ontology imported by any imported ontology, etc. So, if an ontology O_1 imports an ontology O_2 and O_2 imports an ontology O_3 , then every model of O_1 must be a model of O_2 , by condition 2 above. But, every model of O_2 must also be a model of O_3 , again by condition 2 above. Consequently, every model of O_1 must be a model of O_3 .

We now proceed with definitions of the WSML-DL semantic notions, which are in line with the semantic notions typically used in Description Logic reasoning [1].

Concept Satisfiability Given a concrete domain scheme \mathfrak{S} , let c be a concept identifier. We say that c is *WSML-DL \mathfrak{S} -satisfiable* with respect to a WSML-DL ontology O if there is a WSML-DL \mathfrak{S} -model \mathcal{I} of O such that $\mathcal{I}_F(c)_{\text{ext}} \neq \emptyset$, i.e., the concept extension of c is not empty.

Ontology Satisfiability Given a concrete domain scheme \mathfrak{S} , a WSML-DL ontology O is *WSML-DL \mathfrak{S} -satisfiable* if O has a WSML-DL \mathfrak{S} -model.

Formula Entailment Given a concrete domain scheme \mathfrak{S} , a WSML-DL ontology O *WSML-DL \mathfrak{S} -entails* a formula ϕ if every WSML-DL \mathfrak{S} -model of O is a WSML-DL \mathfrak{S} -model of ϕ .

Ontology Entailment Given a concrete domain scheme \mathfrak{S} , a WSML-DL ontology O_1 *WSML-DL \mathfrak{S} -entails* a WSML-DL ontology O_2 if every WSML-DL \mathfrak{S} -model of O_1 is a WSML-DL \mathfrak{S} -model of O_2 .

3.1.4 Stable Models for Core, Flight, and Rule

The semantics of WSML Core, Flight, and Rule is defined on the Stable Model Semantics [11] for logic programs with negation.

It is the case that every consequence under the Well-Founded Semantics is a consequence under the Stable Model Semantics [10]; we explicitly allow implementations to use the Well-Founded Semantics as an approximation to the Stable Model Semantics of WSML-Rule for the task of query answering.

Since all WSML-Flight rules must be *locally stratified*, the semantics corresponds with the perfect model semantics [19] and the Well-Founded Semantics [10] for WSML-Flight ontologies.

We first define the notion of a Herbrand model, in which each ground term is interpreted as itself, and the notion of a *minimal* Herbrand model. We then define the notion of a stable model, which is based on the *reduct* (i.e., removal of negation) of the grounding of a set of WSML rules. Based on the notion of stable models we define satisfiability and entailment for WSML-Core, Flight, and Rule ontologies.



3.1.4.1 Herbrand Models

We first define a specific kind of WSML interpretations and models, namely Herbrand interpretations and models. In these interpretations, the domain of interpretation consists of all ground terms and every ground term is interpreted as itself.

We define Herbrand interpretations relative to a concrete domain scheme $\mathfrak{S} = \langle U^{\mathfrak{S}}, \mathcal{F}^{\mathfrak{S}}, \mathcal{D}^{\mathfrak{S}}, \mathcal{P}^{\mathfrak{S}}, \mathcal{D}^{\mathfrak{S}}, \cdot^{\mathfrak{S}} \rangle$. With \mathcal{D} we denote the set of all data values in \mathfrak{S} : $\mathcal{D} = \bigcup \{d^{\mathfrak{S}} \mid d \in D^{\mathfrak{S}}\}$.

When defining the Herbrand universe we consider symbols from the language for defining abstract terms, but concrete data values are represented by themselves. This is necessary because a concrete data value may have several syntactical representations (e.g., $1.0=1$).

The *Herbrand \mathfrak{S} -universe* of a theory Φ is the union of the set of all ground terms that can be formed from the constant and function symbols in Φ that are not data values and the set of all values $t^{\mathfrak{S}}$ in \mathcal{D} whose lexical representation t occurs in Φ .

A Herbrand \mathfrak{S} -interpretation $\mathcal{I} = \langle U, \prec_U, \in_U, U^D, \mathcal{I}_F, \mathcal{I}_P, \mathcal{I}_{hv}, \mathcal{I}_{it}, \mathcal{I}_{ot} \rangle$ is a \mathfrak{S} -interpretation such that

- U is the Herbrand \mathfrak{S} -universe of Φ and
- for every ground term t that is not a data value, $t^{\mathcal{I}} = t'$, where t' is obtained from t by replacing every data value d in t with $d^{\mathfrak{S}}$.

We can view a Herbrand \mathfrak{S} -interpretation \mathcal{I} as a set of ground atomic formulas, namely that set of ground atomic formulas that is satisfied by \mathcal{I} : $\{\alpha \mid \mathcal{I} \models \alpha\}$. In the following, we will use the symbol \mathcal{I} to denote both the interpretation and the corresponding set of ground atomic formulas.

We can now define the notion of a *minimal Herbrand \mathfrak{S} -model*. Given a concrete domain scheme \mathfrak{S} , a Herbrand \mathfrak{S} -interpretation \mathcal{I} is a *minimal Herbrand \mathfrak{S} -model* of a theory Φ if $\mathcal{I} \models \Phi$ and for every Herbrand \mathfrak{S} -interpretation \mathcal{I}' holds that if $\mathcal{I}' \models \Phi$, then $\mathcal{I} \subseteq \mathcal{I}'$.

3.1.4.2 Stable Models

It turns out that a WSML-Core or Flight theory has at most one Herbrand model, which can be computed using a fixpoint operator in the usual way [17]. However, there is no known computational procedure for finding minimal Herbrand models in case there are multiple models due to the presence of negation in rules. The Stable Model Semantics provides a procedure for finding minimal models, but not every minimal model is a stable model. However, as we have mentioned before, stable models correspond to extensions in default logic, and therefore adequately capture our desired notion of default negation.

The computation of a stable model roughly works as follows:

1. Guess a \mathfrak{S} -model \mathcal{I} of Φ ,
2. create the *reduct* of the grounding of Φ , which “evaluates” the negation in Φ according to \mathcal{I} , and
3. if \mathcal{I} is the minimal Herbrand \mathfrak{S} -model of Φ , then it is a stable model.



The *grounding* of a set of WSML formulas Φ , denoted $gr(\Phi)$, is the union of all possible ground instantiations of Φ , obtained by

- replacing each abstract (resp., concrete) variable in a formula $\phi \in \Phi$ with a ground (resp., ground concrete) term in the Herbrand \mathfrak{S} -universe and
- replacing each data value d in ϕ with the corresponding concrete data value $d^{\mathfrak{S}}$,

for each formula $\phi \in \Phi$.

Following [12], the *reduct* of Φ with respect to a \mathfrak{S} -interpretation \mathcal{I} , denoted $\Phi^{\mathcal{I}}$, is obtained from $gr(\Phi)$ by

- deleting each formula r with a *not* c in the antecedent such that $c \in \mathcal{I}$ and
- deleting *not* c from the antecedent of every remaining formula r .

We are now ready to define the notion of stable \mathfrak{S} -model.

Given a concrete domain scheme \mathfrak{S} , a Herbrand \mathfrak{S} -interpretation \mathcal{I} is a *stable \mathfrak{S} -model* of a WSML-Core, Flight, or Rule theory Φ if \mathcal{I} is a minimal Herbrand \mathfrak{S} -model of $\Phi^{\mathcal{I}}$.

Example 2. Consider the following ground WSML-Rule theory:

$$\begin{aligned} \text{not } p(1.0) &\supset p(2.0) \\ \text{not } p(2) &\supset p(1) \\ p(1.00) &\supset p(3.0) \\ p(2.00) &\supset p(3) \end{aligned}$$

Given any WSML-compliant domain scheme \mathfrak{S} , the theory has two stable \mathfrak{S} -models:

$$\begin{aligned} \mathcal{I}_1 &= \{p(1), p(3)\} \\ \mathcal{I}_2 &= \{p(2), p(3)\} \end{aligned}$$

Observe that the decimals and integer 1.0, 1.00, 1 are all syntactical representations of the integer value 1. Likewise for 2.0, 2.00, 2 and 3.0, 3. The stable models only contain the values corresponding to be syntactical representations.

We leave it as an exercise to the reader to verify that \mathcal{I}_1 and \mathcal{I}_2 are indeed the stable models of the theory.

3.1.4.3 Semantics of WSML-Core, Flight, and Rule Ontologies

We are now ready to define the semantics of WSML-Core, Flight, and Rule ontologies. We are interested in (1) satisfiability of an ontology and (2) entailment of ground formulas.

To deal with imported ontologies we generally followed the same scheme as with WSML-DL.

Note that we have not addressed the semantics of *ot*-molecules (*ofType*) so far. We address the semantics of such molecules in the definition of stable models of ontologies. Specifically, stable models may not satisfy the following formula:

$$\exists c, p, d, a, b (c[p \text{ ot } d] \wedge a : c \wedge a[p \text{ hv } b] \wedge \text{not } b : d) \quad (3.6)$$



which is true in a model if there is a concept c with an attribute p , which is ofType d , there is an instance a of c that has a value b for the attribute p and it is not known that b is an instance of d . In other words, if (3.6) is satisfied, some ofType constraint is violated.

Every WSML ontology O that imports a set of ontologies $\{O_1, \dots, O_n\}$ has an *imports-closed* corresponding theory Φ^\cup , which is defined as follows: $\Phi^\cup = \Phi \cup \Phi_1 \cup \dots \cup \Phi_n$, where Φ is the theory that corresponds to O and Φ_1, \dots, Φ_n are the imports-closed theories of the ontologies O_1, \dots, O_n , respectively.

Given a concrete domain scheme \mathfrak{S} , an interpretation \mathcal{I} is a *stable \mathfrak{S} -model* of a WSML-Core, Flight, or Rule ontology O if

1. \mathcal{I} is a stable \mathfrak{S} -model of the imports-closed theory Φ corresponding to O and
2. \mathcal{I} does not satisfy (3.6).

Satisfiability Given a concrete domain scheme \mathfrak{S} , a WSML-Core, Flight, or Rule ontology O is *\mathfrak{S} -satisfiable* if O has a stable \mathfrak{S} -model.

Entailment Given a concrete domain scheme \mathfrak{S} , a satisfiable WSML-Core, Flight, or Rule ontology O *\mathfrak{S} -entails* a ground atomic formula α if for every stable \mathfrak{S} -model \mathcal{I} of O holds that $\mathcal{I} \models \alpha$.

Example 3. Consider the theory in Example 2. Since the theory as a stable model, it is satisfiable. Additionally, the theory entails $p(3)$, $p(3.0)$, $p(3.00)$, $p(3.000)$, etc.; recall that $3, 3.0, 3.00, 3.000, \dots$ are all syntactical representations of the integer 3. The theory does not entail $p(1)$ or $p(2)$, since these are both not satisfied in every stable model of the theory.

We denote satisfiability and entailment in a given variant x by \models_x .

3.1.5 WSML Full

Any reasonable semantics for WSML Full must fulfill three conditions:

- Any semantics for WSML Full must define an entailment relation \models_{Full} between WSML Full ontologies and WSML Full formulas;
- if Φ and ϕ are a WSML DL ontology and formula, respectively, then whenever $\Phi \models_{DL} \phi$ holds, $\Phi \models_{Full} \phi$ must hold; and
- if Φ and ϕ are a WSML Rule ontology and rule body without free variables, respectively, then whenever $\Phi \models_{Rule} \phi$ holds, $\Phi \models_{Full} \phi$ must hold.

We note that the proposal in [6, 5] fulfills these conditions.



3.2 Combination with RDFS and OWL DL

This section is structured as follows. We first extend the notion of \mathfrak{S} -model with RDFS ontologies, based on the RDFS model theory [13]. We then extend the notions of WSML-DL \mathfrak{S} -model and \mathfrak{S} -model with OWL DL ontologies (resp. the DLP subset), based on the OWL DL model theory [18, Section 3]. Finally, we define the notions of satisfiability and entailment for combinations of WSML-DL with OWL DL and combinations of WSML-Rule with RDFS and (the DLP subset of) OWL DL, respectively.

3.2.1 Combination with RDFS

The RDF semantics specification [13] defines the semantics of RDFS with data types using the notions of D -interpretation and D -entailment for RDFS with datatypes. We define the semantics of combinations of WSML and RDFS ontologies by combining WSML interpretations with D -interpretations.

To this end, we first establish a correspondence between concrete domain schemes and datatype maps, after which we define the notion of combined WSML-RDF interpretations. Note that concrete domain schemes are more general than datatype maps, because they include built-in functions and predicates.

3.2.1.1 Data Types, Datatype Maps, and Concrete Domain Schemes

The symbol D in D -interpretation and D -entailment denotes a *datatype map*, which provides the datatypes that are to be considered in a particular entailment question. So, their purpose is similar to that of concrete domain schemes.

We proceed with the formal definitions of datatypes and datatype maps, following [13, Section 5], and define the notion of compatibility between concrete domain schemes and datatype maps.

Datatypes define sets of concrete data values (e.g., strings and integers), along with their lexical representations. Formally, a *datatype* d consists of

- a *lexical space* L^d , which is a set of Unicode character strings (e.g., “0”, “1”, “01”, \dots , in the case of the `xsd#integer` datatype), which are the lexical representations of the data values,
- a *value space* V^d , which is a set of values (e.g., the numbers 0, 1, 2, \dots , in the case of the `xsd#integer` datatype), and
- a *lexical-to-value mapping* $L2V^d$, which is a mapping from the lexical space to the value space (e.g., $\{“0” \mapsto 0, “1” \mapsto 1, “01” \mapsto 1, \dots\}$, for the `xsd#integer` datatype).

A *datatype map* D is a partial mapping from IRIs to datatypes. With $dom(D)$ we denote the domain of D and with $ran(D)$ we denote the range of D . So, $dom(D)$ is the set of datatype identifiers and $ran(D)$ is the set of datatypes.

We assume that if any two datatype maps D_1 and D_2 are both defined on a datatype identifier d , i.e., they both interpret the same datatype identifier, then they map d to the same datatype: if $d \in dom(D_1)$ and $d \in dom(D_2)$,



then $D_1(d) = D_2(d)$. This assumption is reasonable, because IRIs are global identifiers; therefore, a single IRI may not identify two different datatypes.

Compatibility between Concrete Domain Schemes and Datatype Maps Intuitively, a concrete domain scheme is compatible with a datatype map if the sets of datatype identifiers are the same, these datatype identifiers can be used to identify the same sets of values, and there is a correspondence between the mappings from the syntax to the semantics.

Formally, we say that a concrete domain scheme $\mathfrak{S} = \langle U^\mathfrak{S}, \mathcal{F}^\mathfrak{S}, \mathcal{D}^\mathfrak{S}, \mathcal{P}^\mathfrak{S}, \mathcal{D}^\mathfrak{S}, \cdot^\mathfrak{S} \rangle$ is *compatible* with a datatype map D if

- $dom(D) = \mathcal{D}^\mathfrak{S}$,
- for each n -ary $d \in \mathcal{D}^\mathfrak{S}$ holds that
 - the range of $d^\mathfrak{S}$ is the same as $V^{D(d)}$, and
 - there is a 1-to-1 correspondence between n -tuples $\langle t_1, \dots, t_n \rangle$, where t_1, \dots, t_n are elementary data values, in the domain of $d^\mathfrak{S}$ and character sequences $s \in L^{D(d)}$ such that $d^\mathfrak{S}(\langle t_1, \dots, t_n \rangle) = L2V^{D(d)}(s)$.

Since we assume that there is no difference in the mapping of datatype identifier D between any two datatype maps, each concrete domain scheme \mathfrak{S} has a unique minimal (wrt. $dom(D)$) compatible datatype map D , which we denote with $D_\mathfrak{S}$.

Consider, for example, a concrete domain scheme \mathfrak{S} that includes the `xsd#date` datatype (i.e., `xsd#date` $\in \mathcal{D}^\mathfrak{S}$). The domain of $D_\mathfrak{S}$ includes `xsd#date` and the lexical-to-value mapping is compatible with `xsd#date` ^{\mathfrak{S}} ; for example, `xsd#date(2007, 7, 6)` is the same as $L2V^{D_\mathfrak{S}}(\text{xsd\#date})(\text{"2007-07-06"})$, namely it is the date “July 6, 2007”.

3.2.1.2 Combined WSML-RDF Interpretations

We first review the notion of D -interpretation, after which we define the notion of a combined WSML-RDF \mathfrak{S} -interpretation, which is used to interpret both WSML ontologies and RDF graphs.

A D -interpretation [13] is a tuple $I = \langle IR, IP, LV, IS, IL, IEXT \rangle$, where IR is a non-empty set, called the domain, $IP \subseteq IR$ is a countable set of properties, $LV \subseteq IR$ is a set of literal values that includes all Unicode character sequences, IS is a mapping from IRIs to elements of IR , IL is a mapping from typed literals (i.e., pairs of Unicode character sequences and datatype IRIs) to elements of IR , and $IEXT$ is an extension function that maps elements in IR to sets of pairs of elements in IR : $IEXT : IP \rightarrow 2^{(IR \times IR)}$. Additionally, every D -interpretation must satisfy a number of conditions, as specified in [13]; these conditions govern the behavior of the RDFS ontology modeling vocabulary and the connection between interpretations and datatype maps.

Given a concrete domain scheme \mathfrak{S} . Let $D_\mathfrak{S}$ be the minimal compatible datatype map. A combined WSML-RDF \mathfrak{S} -interpretation is a pair $\langle \mathcal{I}, I \rangle$, where $\mathcal{I} = \langle U, \prec_U, \in_U, U^D, \mathcal{I}_F, \mathcal{I}_P, \mathcal{I}_{hv}, \mathcal{I}_{it}, \mathcal{I}_{ot} \rangle$ is a WSML \mathfrak{S} -interpretation and $I = \langle IR', IP, LV, IS, IL, IEXT \rangle$ is a $D_\mathfrak{S}$ -interpretation, such that the following conditions hold:

1. $IR' = U \cup U^D$,



2. $LV = U^D$,
3. $IP \supseteq \{p \mid \exists s, o. \langle s, o \rangle \in \mathbf{I}_{\rightarrow}(p)\}$,
4. $IS(t) = \mathbf{I}_F(t)$ for every IRI t ,
5. $IEXT = \mathbf{I}_{hv}$,
6. $IEXT(IS(rdf:type)) = \in_U$, and
7. $IEXT(IS(rdfs:subClassOf)) = \{\langle a, b \rangle \mid a, b \in U \ \& \ a \preceq b \ \& \ a \in_U IS(rdfs:Class) \ \& \ b \in_U IS(rdfs:Class)\}$.

Note that the interpretation of (well-)typed literals in I corresponds with the interpretation of data values in \mathcal{I} due to the compatibility of \mathfrak{S} and $D_{\mathfrak{S}}$. Likewise, the interpretation of plain literals without language tags corresponds with the interpretation of strings, due to the way plain literals are interpreted in RDF [13]. Note also that there is no representation in WSML of ill-typed literals (e.g., “ a ” \sim $xsd:integer$).

Conditions 1 and 2 ensures that the domains under consideration are the same. Condition 3 ensures that the set of properties in the RDF interpretation includes at least those elements that are used as properties in the WSML interpretation. Condition 4 ensures that all IRIs are interpreted in the same way.

Due to condition 5, there is a correspondence between RDF triples of the form $\langle s, p, o \rangle$ and WSML molecules of the form $s[p\ hv\ o]$. Likewise, there is a correspondence between RDF triples of the form $\langle a, rdf:type, c \rangle$ and WSML molecules of the form $a:c$ and a correspondence between RDF triples of the form $\langle c, rdfs:subClassOf, d \rangle$ and WSML molecules of the form $c::d$, by the conditions 6 and 7. Namely, whenever $\langle c, rdfs:subClassOf, d \rangle$, then $c::d$. Note that it is not necessarily the case that if $c::d$, then $\langle c, rdfs:subClassOf, d \rangle$, because $c::d$ holds whenever $c = d$, whereas $\langle c, rdfs:subClassOf, d \rangle$ may not hold even if $c = d$; in RDFS, the subclass relation may only hold between elements which have the type $rdfs:Class$.

Before defining models, and entailment we first define interpretations of combinations with OWL DL. We then define satisfiability and entailment of combinations of the three kinds of ontologies.

3.2.2 Combination with OWL DL

We now extend the notion of combined WSML-RDF interpretations to include OWL DL interpretations, leading to the notion of WSML-RDF-OWL interpretation. The OWL DL interpretations are used for interpreting OWL ontologies, and correspondence between the WSML and OWL interpretations is defined through a number of conditions, analogous to the correspondence between RDF and WSML interpretations.

We note that the notion of OWL DL interpretation we use here corresponds to the notion of *abstract OWL interpretation* [18, Section 3.1].

Given a datatype map D , an abstract OWL interpretation with respect to D is a tuple $\mathfrak{J} = \langle R, EC, ER, L, S, LV \rangle$, where



- R is a nonempty set,
- $LV \subseteq R$ is the set of literal values,
- EC maps class and datatype identifiers to sets,
- ER maps properties to binary relations,
- L maps typed literals to literal values, and
- S maps IRIs to elements of R .

Additionally, every abstract OWL interpretation has to satisfy a number of conditions, as specified in [18, Section 3].

Given a concrete domain scheme \mathfrak{S} and the minimal compatible datatype map $D_{\mathfrak{S}}$, a *combined WSML-RDF-OWL \mathfrak{S} -interpretation* is a tuple $\langle \mathcal{I}, I, \mathcal{J} \rangle$, where $\mathcal{I} = \langle U, \prec_U, \in_U, U^D, \mathcal{I}_F, \mathcal{I}_P, \mathcal{I}_{hv}, \mathcal{I}_{it}, \mathcal{I}_{ot} \rangle$ is a WSML interpretation that conforms with \mathfrak{S} , $I = \langle IR, IP, IS, IEXT \rangle$ is a $D_{\mathfrak{S}}$ -interpretation, and $\mathcal{J} = \langle R, EC, ER, L, S, LV \rangle$ is an OWL DL interpretation with respect to $D_{\mathfrak{S}}$, such that $\langle \mathcal{I}, I \rangle$ is a combined WSML-RDF \mathfrak{S} -interpretation and the following conditions hold:

1. $R = U \cup U^D$,
2. $LV = U^D$,
3. for every concept or datatype identifier c , $EC(c) = \{k \mid k \in U \cup U^D \ \& \ k \in_U \mathcal{I}_F(c)\}$,
4. for every property identifier p , $ER(p) = \{\langle a, b \rangle \mid a, b \in U \cup U^D \ \& \ \langle a, b \rangle \in \mathcal{I}_{\rightarrow}(\mathcal{I}_F(p))\}$, and
5. $S(t) = \mathcal{I}_F(t)$ for every IRI t .

Conditions 1 and 2 ensure that all ontologies talk about the same domain. Conditions 3 and 4 ensure that classes of properties are interpreted the same way. Finally, condition 5 ensures that IRIs are interpreted in the same way.

3.2.3 Satisfiability and Entailment of Combinations

We are now ready to define the semantics of WSML ontologies that import RDFS and/or OWL DL ontologies and of combinations of WSML, RDFS, and OWL DL ontologies. Such combinations may arise, for example, if several different ontologies are imported in a Web service description.

As before we distinguish between the semantics of the WSML-DL variant, on the one hand, and the WSML-Core, Flight, and Rule variants, on the other. The notions of model, satisfiability, and entailment are extensions of the corresponding notions for WSML ontologies defined above.



3.2.3.1 WSML-DL Semantics of Combinations

Recall that, when considering the WSML-DL variant, combinations with RDFS ontologies are not allowed. Therefore, we only consider WSML-DL and OWL DL ontologies in the combination.

In the definition of the semantics of WSML-DL ontologies, in Section 3.1.3, we defined the notions of models, satisfiability, and entailment for WSML-DL ontologies that import other WSML-DL ontologies. We now consider both WSML-DL and OWL DL ontologies that possibly import both WSML-DL and OWL DL ontologies.

Given a concrete domain scheme \mathfrak{S} , a combined WSML-RDF-OWL interpretation $\langle \mathcal{I}, I, \mathcal{J} \rangle$ is a *DL \mathfrak{S} -model* of a WSML-DL or OWL DL ontology O if

1. \mathcal{I} is WSML-DL interpretation,
2. in case O is a WSML-DL ontology, \mathcal{I} is a WSML-DL \mathfrak{S} -model of a theory Φ that corresponds to O ,
3. in case O is an OWL DL ontology, \mathcal{J} satisfies O with respect to $D_{\mathfrak{S}}$ according to [18, Section 3.4], and
4. $\langle \mathcal{I}, I, \mathcal{J} \rangle$ is a DL \mathfrak{S} -model of every ontology imported by O .

We are now ready to define the semantic notions of WSML-DL, analogous to the notions defined in Section 3.1.3.

Concept Satisfiability Given a concrete domain scheme \mathfrak{S} , let c be a concept identifier. We say that c is *DL \mathfrak{S} -satisfiable* with respect to a set of WSML-DL and OWL DL ontologies \mathbf{O} if there is a combined WSML-RDF-OWL interpretation $\langle \mathcal{I}, I, \mathcal{J} \rangle$ that is a model of every $O \in \mathbf{O}$ such that $\mathcal{I}_F(c)_{\text{cext}} \neq \emptyset$, i.e., the concept extension of c is not empty.

Ontology Satisfiability Given a concrete domain scheme \mathfrak{S} , a set of WSML-DL and OWL DL ontologies \mathbf{O} is *DL \mathfrak{S} -satisfiable* if there is a combined WSML-RDF-OWL interpretation $\langle \mathcal{I}, I, \mathcal{J} \rangle$ that is a model of every $O \in \mathbf{O}$.

Formula Entailment Given a concrete domain scheme \mathfrak{S} , a set of WSML-DL and OWL DL ontologies \mathbf{O} *DL \mathfrak{S} -entails* a formula ϕ if for every combined WSML-RDF-OWL interpretation $\langle \mathcal{I}, I, \mathcal{J} \rangle$ that is a model of every $O \in \mathbf{O}$ holds that \mathcal{I} is a WSML-DL \mathfrak{S} -model of ϕ .

Ontology Entailment Given a concrete domain scheme \mathfrak{S} , a set of WSML-DL and OWL DL ontologies \mathbf{O} *DL \mathfrak{S} -entails* a WSML-DL or OWL DL ontology O if for every combined WSML-RDF-OWL interpretation $\langle \mathcal{I}, I, \mathcal{J} \rangle$ that is a model of every $O' \in \mathbf{O}$ holds that $\langle \mathcal{I}, I, \mathcal{J} \rangle$ is a DL \mathfrak{S} -model of O .

3.2.3.2 WSML-Core, Flight and Rule Semantics of Combinations

As we did for WSML ontologies, we define the semantics of WSML-Core, Flight, and Rule combinations using a notion of stable models. The domains of these stable models are somewhat extended, compared with the stable models of WSML ontologies, in order to account for the blank nodes in RDF graphs.



So, we essentially *Skolemize* the blank nodes by replacing them with constant symbols.

We note that the semantics of these three variants is only defined for OWL DL ontologies that fall within the DLP fragment, i.e., those ontologies that are equivalent to sets of equality-free Horn formulas, and are hence essentially negation-free rules.

We first extend the notion of Herbrand models to combinations. We then define the notion of a stable model of a combination.

Herbrand Models of Combinations Recall that we define Herbrand interpretations relative to a concrete domain scheme $\mathfrak{S} = \langle U^{\mathfrak{S}}, \mathcal{F}^{\mathfrak{S}}, \mathcal{D}^{\mathfrak{S}}, \mathcal{P}^{\mathfrak{S}}, \mathcal{D}^{\mathfrak{S}}, \cdot^{\mathfrak{S}} \rangle$. Recall also that \mathcal{D} denotes the set of all data values in \mathfrak{S} : $\mathcal{D} = \bigcup \{d^{\mathfrak{S}} \mid d \in D^{\mathfrak{S}}\}$.

In the remainder of this section, the symbol \mathbf{O} denotes a set of WSML, RDFS, and OWL DL ontologies. The *set of constant symbols* of \mathbf{O} is the set of IRIs, blank nodes, literals, and data values in every ontology in \mathbf{O} (we assume anonymous identifiers have been replaced with IRIs, as described in Section 3.1.4). The set of function symbols of \mathbf{O} is the set of all symbols occurring in a function position in any WSML ontology in \mathbf{O} . The *set of ground terms* of \mathbf{O} is the set of all ground terms that can be formed from the constant and function symbols of \mathbf{O} .

The *Herbrand \mathfrak{S} -universe* of \mathbf{O} is the union of set of ground terms of \mathbf{O} that are not data values and the set of values $t^{\mathfrak{S}}$ in \mathcal{D} whose lexical representation t occurs in any of the ontologies in \mathbf{O} .

A Herbrand WSML-RDF-OWL \mathfrak{S} -interpretation $\langle \mathcal{I}, I, \mathcal{J} \rangle$ of \mathbf{O} is a WSML-RDF-OWL \mathfrak{S} -interpretation where

- the domain of \mathcal{I}, U , is the Herbrand \mathfrak{S} -universe of \mathbf{O} and
- for every ground term t of \mathbf{O} , $t^{\mathcal{I}} = t'$, where t' is obtained from t by replacing every data value d in t with $d^{\mathfrak{S}}$.

Recall the conditions on combined WSML-RDF-OWL interpretations in Sections 3.2.1 and 3.2.2. These conditions ensure that the parts of the domains of RDF and OWL interpretations that are not literal values correspond to the domain of the WSML interpretation \mathcal{I} . Therefore, these domains correspond to the Herbrand \mathfrak{S} -universe as well. Additionally, due to the fact that IRIs are interpreted the same in all three interpretations, all ground terms are interpreted in the same way in all three interpretations.

We are now ready to define the notions of model and minimal model. As before, a model of an ontology is required to be a model of every imported ontology.

A set of WSML, RDFS, and OWL DL ontologies \mathbf{O} is *imports-closed* if for every ontology O imported by any ontology O' in \mathbf{O} holds that $O \in \mathbf{O}$.

Now, a Herbrand WSML-RDF-OWL \mathfrak{S} -interpretation $\langle \mathcal{I}, I, \mathcal{J} \rangle$ is a *\mathfrak{S} -model* of an imports-closed set of ontologies \mathbf{O} if

- for every WSML ontology $O \in \mathbf{O}$, \mathcal{I} is a \mathfrak{S} -model of the theory Φ corresponding to O ,
- for every RDFS ontology $O \in \mathbf{O}$, I satisfies O , and
- for every OWL DL ontology $O \in \mathbf{O}$, \mathcal{J} satisfies O .



We can now define the notion of a *minimal Herbrand \mathfrak{S} -model*. A Herbrand WSML-RDF-OWL \mathfrak{S} -interpretation $\langle \mathcal{I}, I, \mathcal{J} \rangle$ is a *minimal Herbrand \mathfrak{S} -model* of an imports-closed set of ontologies \mathbf{O} if

- $\langle \mathcal{I}, I, \mathcal{J} \rangle$ is a \mathfrak{S} -model of \mathbf{O} and
- whenever a Herbrand \mathfrak{S} -interpretation $\langle \mathcal{I}', I', \mathcal{J}' \rangle$ is an \mathfrak{S} -model of \mathbf{O} and $\mathcal{I}' \subseteq \mathcal{I}$, then $\mathcal{I}' = \mathcal{I}$.

Stable Models of Combinations The definition of stable models of combinations is similar to the definition of stable models for WSML ontologies. Likewise, we guess a model, “evaluate” the grounding of a combination by computing the reduct, and check whether the model we guessed is the minimal Herbrand model of the reduct.

It turns out that in a set of WSML, RDFS, and OWL DL ontologies we only need to ground the WSML ontologies, because the RDFS and OWL DL ontologies do not contain negation (recall that the OWL DL ontologies are required to be in the DLP fragment, which is negation-free).

The *grounding* of \mathbf{O} , denoted $gr(\mathbf{O})$, is obtained from \mathbf{O} by replacing every WSML ontology $O \in \mathbf{O}$ by $gr(O)$, which is the union of all possible ground instantiations of the corresponding WSML theory Φ , obtained by

- replacing each abstract (resp., concrete) variable in a formula $\phi \in \Phi$ with a ground (resp., ground concrete) term in the Herbrand \mathfrak{S} -universe of \mathbf{O} and
- replacing each data value d in ϕ with the corresponding concrete data value $d^{\mathfrak{S}}$,

for each formula $\phi \in \Phi$.

The *reduct* of \mathbf{O} with respect to a combined Herbrand \mathfrak{S} -interpretation $\langle \mathcal{I}, I, \mathcal{J} \rangle$, denoted $\mathbf{O}^{\langle \mathcal{I}, I, \mathcal{J} \rangle}$, is obtained from $gr(\mathbf{O})$ by, for each WSML ontology $O \in gr(\mathbf{O})$,

- deleting each formula r with a *not* c in the antecedent such that $c \in \mathcal{I}$ and
- deleting *not* c from the antecedent of every remaining formula r .

Given a concrete domain scheme \mathfrak{S} , a Herbrand WSML-RDF-OWL \mathfrak{S} -interpretation $\langle \mathcal{I}, I, \mathcal{J} \rangle$ is a *stable \mathfrak{S} -model* of an imports-closed set of ontologies \mathbf{O} if $\langle \mathcal{I}, I, \mathcal{J} \rangle$ is a minimal Herbrand \mathfrak{S} -model of $\mathbf{O}^{\langle \mathcal{I}, I, \mathcal{J} \rangle}$.

Using the notion of stable \mathfrak{S} -model we define the semantic notions for WSML-Core, Flight, and Rule combinations.

Satisfiability Given a concrete domain scheme \mathfrak{S} , an imports-closed set of ontologies \mathbf{O} is *\mathfrak{S} -satisfiable* if \mathbf{O} has a stable \mathfrak{S} -model.

Entailment Given a concrete domain scheme \mathfrak{S} , an imports-closed set of ontologies \mathbf{O} *\mathfrak{S} -entails* a ground atomic formula α if for every stable \mathfrak{S} -model $\langle \mathcal{I}, I, \mathcal{J} \rangle$ of \mathbf{O} holds that $\mathcal{I} \models \alpha$.



3.3 Nonfunctional Property Semantics

The central notion for nonfunctional properties is the *value* of the property. The nonfunctional properties semantics described in this section defines the values of a specific nonfunctional property.

Given a logical expression logExp , a variable substitution θ is a mapping from free variables in logExp , denoted $\text{var}(\text{logExp})$, to identifiers: $\theta : \text{var}(\text{logExp}) \rightarrow \mathbf{Id}$. With $\text{logExp}\theta$ we denote the application of θ to logExp , i.e. the replacement of every free variable x in logExp with $\theta(x)$.

Let $\langle \dots, \mathbf{ontID}, \dots, \{ \dots, \mathit{nfp}, \dots \}, \dots \rangle_x$ be a WSML Web service, goal, capability, interface or mediator with the WSML variant var , where \mathbf{ontID} is the set of imported ontologies and $\mathit{nfp} = \langle \text{name}, \text{value}, \mathbf{logExp} \rangle_{\mathit{nfp}}$ is a nonfunctional property, let \mathbf{O} be the imports-closed set of ontologies induced by \mathbf{ontID} , and let Θ be the set of variable substitutions such that for every $\theta \in \Theta$ it holds that for all $\text{logExp} \in \mathbf{logExp}$, $\mathbf{O} \models_{\text{var}} \text{logExp}\theta$. Then, an identifier $i \in \mathbf{Id}$ is a *value* of name if there is a $\theta \in \Theta$ such that $i = \theta(\text{value})$.

Observe that in case value is an identifier, the logical expressions functions as a filter: if all of the logical expressions are entailed by the ontologies, value is a value of the property.

So, the logical expressions of the nonfunctional property can be seen as queries over the ontologies, and the query answers are projected onto value.

3.4 Choreography Semantics

The semantics of choreographies is defined in terms of *choreography runs*, which are sequences of *states*, which are set of WSML facts. Given a start state, which can be seen as the user input, a choreography has a number of associated choreography runs; this set of associated choreography runs is uniquely determined by the start state, the transition rules, and the imported ontologies of the choreography.

The set of ontologies \mathbf{O}_C of a choreography $C = \langle \text{name}, \mathbf{ann}, \mathbf{ontID}, \mathbf{medID}, \text{signature}, \mathbf{rule} \rangle_{\text{chor}}$ is the imports-closed set of ontologies induced by \mathbf{ontID} . The WSML *variant* of the choreography is variant of \mathbf{O}_C .

A *choreography state* S of the choreography C is a set of ground atoms of *variant*.

A choreography state S is *consistent* with a set of ontologies \mathbf{O}_C if $\mathbf{O}_C \cup S \not\models_{\text{variant}} \perp$.

An *update set* U is a tuple $U = \langle \mathbf{A}, \mathbf{D} \rangle$, where the add set \mathbf{A} and the delete set \mathbf{D} are sets of ground atoms of *variant*. An update set $U = \langle \mathbf{A}, \mathbf{D} \rangle$ is *consistent* if $\mathbf{A} \cap \mathbf{D} = \emptyset$.

The *application* of an update set $U = \langle \mathbf{A}, \mathbf{D} \rangle$ to a state S , denoted S^U , is defined as: $S^U = S \setminus \mathbf{D} \cup \mathbf{A}$.

An update set U is consistent with the state S , with respect to a set of ontology \mathbf{O} , if U is consistent and S^U is consistent with \mathbf{O} .

An update set U is *associated with* a choreography $C = \langle \text{name}, \mathbf{ann}, \mathbf{ontID}, \mathbf{medID}, \text{signature}, \mathbf{rule} \rangle_{\text{chor}}$ and a state S if $U = \pi_C(\mathbf{rule}, S)$, with π_C as defined in Table 3.1.



$$\begin{aligned}
\pi_C(\mathbf{rule}, S) &= \bigcup_{\mathbf{rule} \in \mathbf{rule}} \pi_C(\mathbf{rule}, S), \\
\pi_C(\langle \alpha \rangle_{add}, S) &= \langle \{\alpha\}, \emptyset \rangle, \\
\pi_C(\langle \alpha \rangle_{delete}, S) &= \langle \emptyset, \{\alpha\} \rangle, \\
\pi_C(\langle \logExp, \mathbf{rule} \rangle_{if}, S) &= \begin{cases} \pi_C(\mathbf{rule}, S) & \mathbf{O}_C \cup \{S\} \models_{variant} \logExp, \\ \emptyset & \text{otherwise,} \end{cases} \\
\pi_C(\langle \mathbf{varId}, \logExp, \mathbf{rule} \rangle_{forall}, S) &= \begin{cases} \pi_C(\bigcup \{ \mathbf{rule} \theta \mid \theta \text{ is a mapping from the} \\ \text{variables in } \mathbf{varId} \text{ to ground terms and} \\ \mathbf{O}_C \cup \{S\} \models_{variant} \logExp \theta \}, S), \\ \emptyset & \text{otherwise,} \end{cases} \\
\pi_C(\langle \mathbf{varId}, \logExp, \mathbf{rule} \rangle_{choose}, S) &= \begin{cases} \pi_C(\mathbf{rule} \theta, S) & \theta \text{ is a mapping from} \\ & \text{the variables in } \mathbf{varId} \\ & \text{to ground terms and} \\ & \mathbf{O}_C \cup \{S\} \models_x \logExp \theta, \\ \emptyset & \text{otherwise,} \end{cases}
\end{aligned}$$

where $\langle \mathbf{A}_1, \mathbf{D}_1 \rangle \cup \langle \mathbf{A}_2, \mathbf{D}_2 \rangle = \langle \mathbf{A}_1 \cup \mathbf{A}_2, \mathbf{D}_1 \cup \mathbf{D}_2 \rangle$ and x is the *variant* under consideration.

Table 3.1: Associated update set

Let $C = \langle \text{name}, \mathbf{ann}, \mathbf{ontID}, \mathbf{medID}, \text{signature}, \mathbf{rule} \rangle_{chor}$ be a choreography, let S be a state, and let U be an update set. and the state

A *choreography run* is a sequence of choreography states $R = \langle S_0, \dots, S_n \rangle$, with $n \geq 0$. A state S is *reachable* in R if $S = S_i$ for some $0 \leq i \leq n$.

A choreography run $R = \langle S_0, \dots, S_n \rangle$ is *terminated* if there is an update set U associated with C and S_n such that either

- S_n^U is not consistent with \mathbf{O}_C or
- $S_n^U = S_n$.

In this case we call S_n the terminating state of R .

Given a start state S , choreography run $R = \langle S_0, \dots, S_n \rangle$ is *valid* for a choreography C if $S_0 = S$ is consistent with \mathbf{O}_C , R is terminated, and for each S_i , with $1 \leq i \leq n$, holds that

- S_i is consistent with \mathbf{O}_C ,
- $S_{i-1} \neq S_i$, and
- there is a consistent update set U , which is associated with C and S_{i-1} , such that $S_i = S_{i-1}^U$.

Given a start state S and a choreography C , a state S_i is *terminating* if there is a valid choreography run R for C such that S_i is the terminating state of R ; S_i is *reachable* if there is a valid choreography run R for C such that S_i is reachable in R .



An atomic formula α is *possible* (resp., *guaranteed*) in C , given a start state S , if $\alpha \in S_i$ for some (resp., every) terminating state S_i .

An atomic formula α is *always possible* (resp., *always guaranteed*) in C if for every possible start state S holds that $\alpha \in S_i$ for some (resp., every) terminating state S_i .

A choreography C_1 is *subsumed by* a choreography C_2 , *relative to a start state* S , if every choreography run R with start state S which is valid for C_1 is valid for C_2 .

A choreography C_1 is *always subsumed by* a choreography C_2 if for every possible state S holds that every choreography run R with start state S which is valid for C_1 is valid for C_2 .

A choreography C is *consistent* given a state S_0 if there exists a valid run $R = \langle S_0, \dots, S_n \rangle$ for C .

C is *always consistent* if for any consistent state S , C is consistent given S .



A The Description Logic $\mathcal{SHIQ}(\mathbf{D})$

The signature $\Sigma = \langle \mathcal{C}, \mathcal{D}, \mathcal{R}_a, \mathcal{R}_c, \mathcal{F}_a, \mathcal{F}_c \rangle$ of a $\mathcal{SHIQ}(\mathbf{D})$ [1] language consists of pairwise disjoint sets of concept (\mathcal{C}), datatype (\mathcal{D}), abstract role (\mathcal{R}_a), concrete role (\mathcal{R}_c), individual (\mathcal{F}_a), and data value (\mathcal{F}_c) identifiers. $\mathcal{SHIQ}(\mathbf{D})$ descriptions are defined as follows, with A a concept identifier, D a datatype identifier, C, C' descriptions, R, R' role identifiers, S, S' abstract role identifiers, U, U' concrete role identifiers, a, b individual identifiers, o a data value identifier, and n a non-negative integer.

$$C, C' \longrightarrow \perp \mid A \mid C \sqcap C' \mid \neg C \mid \geq nS.C \mid \geq nU.D \mid \leq nS.C \mid \leq nU.D$$

A $\mathcal{SHIQ}(\mathbf{D})$ ontology is a set of axioms of the following forms.

$$C \sqsubseteq C' \mid S \sqsubseteq S' \mid U \sqsubseteq U' \mid S \equiv S'^- \mid (S)^+ \mid C(a) \mid S(a, b) \mid U(a, o) \mid a = b \mid a \neq b$$

Additionally, we have that in *number restrictions* $\geq nS.C$ and $\leq nS.C$, S has to be *simple*, i.e., S and its sub-roles may not be transitive (with $(S)^+$ denoting transitivity).

We present the semantics of $\mathcal{SHIQ}(\mathbf{D})$ through a translation to first-order logic;¹ for a description of the correspondence between the DL semantics and FOL, see e.g. [1]. The function π maps $\mathcal{SHIQ}(\mathbf{D})$ descriptions to first-order logic formulas with one free variable, X .

Mapping concepts to FOL	
$\pi(A, X)$	$A(X)$
$\pi(\perp, X)$	\perp
$\pi(C \sqcap C', X)$	$\pi(C, X) \wedge \pi(C', X)$
$\pi(\neg C, X)$	$\neg \pi(C, X)$
$\pi(\geq nS.C, X)$	$\exists_a y_1, \dots, y_n (\bigwedge_{1 \leq i \leq n} (S(X, y_i) \wedge \pi(C, y_i)) \wedge \bigwedge_{i \neq j} \neg y_i = y_j)$
$\pi(\leq nS.C, X)$	$\forall_a y_1, \dots, y_{n+1} (\bigwedge_{1 \leq i \leq n+1} (S(X, y_i) \wedge \pi(C, y_i)) \supset \bigvee_{i \neq j} y_i = y_j)$
$\pi(\geq nU.D, X)$	$\exists_c y_1, \dots, y_n (\bigwedge_{1 \leq i \leq n} (U(X, y_i) \wedge \pi(D, y_i)) \wedge \bigwedge_{i \neq j} \neg y_i = y_j)$
$\pi(\leq nU.D, X)$	$\forall_c y_1, \dots, y_{n+1} (\bigwedge_{1 \leq i \leq n+1} (U(X, y_i) \wedge \pi(D, y_i)) \supset \bigvee_{i \neq j} y_i = y_j)$

where y_1, \dots, y_{n+1} are new variables

The correspondence between $\mathcal{SHIQ}(\mathbf{D})$ axioms and first-order logic formulas is as follows.

DL axiom	FOL equivalent	DL axiom	FOL equivalent
$C \sqsubseteq C'$	$\forall_a x (\pi(C, x) \supset \pi(C', x))$	$C(a)$	$\pi_y(C, a)$
$S \sqsubseteq S'$	$\forall_a x, y (S(x, y) \supset S'(x, y))$	$S(a, b)$	$S(a, b)$
$U \sqsubseteq U'$	$\forall_a x (\forall_c y (U(x, y) \supset U'(x, y)))$	$U(a, o)$	$U(a, o)$
$S \equiv S'^-$	$\forall_a x, y (S(x, y) \equiv S'(y, x))$	$a = b$	$a = b$
$(S)^+$	$\forall_a x, y, z (S(x, y) \wedge S(y, z) \supset S(x, z))$	$a \neq b$	$\neg(a = b)$

¹It is easy to verify that the semantics we use for concrete domains corresponds to the semantics usually considered in Description Logics when considering $\mathcal{SHIQ}(\mathbf{D})$ (e.g. [2]).



B WSML Layering

We consider two approaches to language layering in WSML. When considering *loose* layering, a variant L_2 is layered on a variant L_1 if, considering an arbitrary theory of L_1 , every L_1 -formula that is a consequence under L_1 semantics, is also a consequence under L_2 semantics. When considering *strict* layering, additionally every L_1 -formula that is a consequence under L_2 semantics must be a consequence under L_1 semantics. Note that loose layering implies strict layering.

It turns out that if we want to guarantee strict language layering in WSML, we must pose certain restrictions on the entailments that are considered; namely, subclass and implies-type statements may not be considered. Additionally, we need to pose restrictions on the antecedents of WSML-Flight and Rule formulas; namely, subclass and implies-type statements may not appear in the antecedents, because they could be used to “simulate” entailment of such statements.

With WSML-Flight⁻ (resp., Rule⁻) we mean the WSML variant obtained from WSML-Flight (resp., Rule) by disallowing :- and it-molecules in the antecedents of formulas (rule bodies). WSML-Core⁻ is the same as WSML-Core.

Admissible entailments are subsets of all formulas of a given WSML variant. The admissible entailments of the WSML variants are defined as follows:

- Core⁻/Flight⁻/Rule⁻: every it- and :-free WSML-(Core/Flight/Rule) ground atomic formula is an admissible entailment;
- Core/Flight/Rule: every WSML-(Core/Flight/Rule) ground atomic formula is an admissible entailment;
- DL: every WSML-DL sentence is an admissible entailment.

Using this notion of admissible entailment, we can now formally define language layering.

Definition 1. *Let L_1, L_2 be two WSML variants. Then, L_2 is loosely layered on top of L_1 , denoted $L_1 \Rightarrow_l L_2$, if for every L_1 theory Φ and admissible entailment of L_1 , α is an L_2 entailment of Φ whenever α is an L_1 entailment of Φ .*

If, in addition, α is an L_1 entailment of Φ whenever α is a L_2 entailment of Φ , L_2 is strictly layered on top of L_1 , denoted $L_1 \Rightarrow_s L_2$.

Lemma 1. *Given a concrete domain scheme \mathfrak{S} . Let Φ be a WSML-Core theory and let α be a ground atomic formula.*

- Φ has a single minimal Herbrand \mathfrak{S} -model iff Φ has a \mathfrak{S} -model and
- α is satisfied in every minimal Herbrand \mathfrak{S} -model of Φ iff α is satisfied in every \mathfrak{S} -model of Φ .

Sketch. Φ can straightforwardly be transformed to an equi-satisfiable first-order theory Φ' by (1) replacing every data value d with its interpretation $d^{\mathfrak{S}}$, (2) replacing molecules with binary and ternary predicates (e.g., replace $a::b$ with



$_subclass(a, b)$), and (3) add axioms that captures the semantic conditions of the subclass and implies-type molecules; these axioms axiomatize the conditions (3.1) and (3.2). Φ' is obviously a Horn logic theory. The lemma follows immediately from the classical results by Herbrand (e.g., [9]). \square

Theorem 1 (WSML Language Layering).

1. $WSML-Core \Rightarrow_l WSML-Flight \Rightarrow_l WSML-Rule$.
2. $WSML-Core \Rightarrow_l WSML-DL$.
3. $WSML-Core^- \Rightarrow_s WSML-Flight^- \Rightarrow_s WSML-Rule^-$.
4. $WSML-Core^- \Rightarrow_s WSML-DL$.

Proof. 1. and 3. follow immediately from the definition of the semantics of WSML-Core, WSML-Flight, and WSML-Rule.

By Lemma 1 we need to consider only a single minimal model for checking satisfiability and ground entailment, even if a theory has multiple models and the domain of the minimal model is partitioned.

The minimal Herbrand model \mathcal{I} of a WSML-Core theory Φ can be straightforwardly transformed into an interpretation \mathcal{I}' that satisfies the conditions (3.4) and (3.5). Since \mathcal{I}' is a Herbrand model and since there is a distinction in the WSML-Core syntax between instance, concept, and attribute identifiers, it is easy to find a partitioning of the domain into instance, attribute, and concept domains. Additionally, by the fact that \mathcal{I} is a minimal model and by the syntactical restrictions on WSML-Core theories, \prec_U is only defined on U^c , \in_U is a relation between U^i and U^c , and $\mathcal{I}_{hv}(u) = \mathcal{I}_{it}(u) = \mathcal{I}_{ot}(u) = \emptyset$ for any $u \in U^c \cup U^i$. Therefore, \mathcal{I}' is a WSML-DL interpretation.

By the syntactical restrictions on WSML-Core theories it is easy to verify that \mathcal{I}' WSML-DL satisfies Φ and that \mathcal{I}' is in fact a minimal WSML-DL model of Φ , establishing loose layering (2.).

Given an it- and ::-free ground atomic formula α , clearly $\mathcal{I} \models \alpha$ iff $\mathcal{I}' \models_{DL} \alpha$. This establishes strict layering between WSML-Core⁻ and WSML-DL (4.). \square

The use of loose layering seems more attractive than strict layering. Indeed, we have chosen to use loose layering for the standard language variants. In addition, the use of loose language layering is common in Semantic Web standards; for example, RDFS is loosely layered on top of RDF, OWL Full is loosely layered on top of RDFS, and OWL DL is loosely layered on top of OWL DL. However, one could imagine scenarios in which strict language layering is more attractive. For example, when directly using a WSML-DL reasoner for reasoning with WSML-Core theories, one needs to be sure that the semantics correspond; otherwise, certain inferences might be incorrect with respect to the WSML-Core semantics. In this case, one needs to restrict oneself to WSML-Core⁻.

Acknowledgements

The author would like to thank to all the members of the WSML working group for their advice and input into this document.



Bibliography

- [1] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2003.
- [2] Franz Baader and Philipp Hanschke. A scheme for integrating concrete domains into concept languages. Technical Report RR-91-10, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH, 1991.
- [3] Paul V. Biron and Ashok Malhotra. Xml schema part 2: Datatypes second edition. Recommendation 28 October 2004, W3C, 2004.
- [4] Alex Borgida. On the relative expressiveness of description logics and predicate logics. *Artificial Intelligence*, 82(1–2):353–367, 1996.
- [5] Jos de Bruijn and Stijn Heymans. WSML ontology semantics. Final Draft d28.3, WSML Working Group, 2007.
- [6] Jos de Bruijn and Stijn Heymans. A semantic framework for language layering in WSML. In *Proceedings of the First International Conference on Web Reasoning and Rule Systems (RR2007)*, Innsbruck, Austria, June 7–8 2007. Springer.
- [7] M. Duerst and M. Suignard. Internationalized resource identifiers (IRIs). RFC 3987, IETF, 2005. <http://www.ietf.org/rfc/rfc3987.txt>.
- [8] Ioan Toma (Ed.). WSML/XML. Working Draft D36 v1.0, WSML, 2008. <http://www.wsmo.org/TR/d36/v1.0/>.
- [9] M. Fitting. *First Order Logic and Automated Theorem Proving (second edition)*. Springer Verlag, 1996.
- [10] Allen Van Gelder, Kenneth Ross, and John S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.
- [11] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert A. Kowalski and Kenneth Bowen, editors, *Proceedings of the Fifth International Conference on Logic Programming*, pages 1070–1080, Cambridge, Massachusetts, 1988. The MIT Press.
- [12] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3/4):365–386, 1991.
- [13] Patrick Hayes. RDF semantics. Technical report, W3C, 2004.
- [14] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Practical reasoning for very expressive description logics. *Logic Journal of the IGPL*, 8(3):239–264, May 2000.



- [15] Michael Kifer, Georg Lausen, and James Wu. Logical foundations of object-oriented and frame-based languages. *JACM*, 42(4):741–843, 1995.
- [16] Graham Klyne and Jeremy J. Carroll. Resource description framework (RDF): Concepts and abstract syntax. Recommendation 10 February 2004, W3C, 2004.
- [17] John W. Lloyd. *Foundations of Logic Programming (2nd edition)*. Springer-Verlag, 1987.
- [18] Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks. OWL web ontology language semantics and abstract syntax. Recommendation 10 February 2004, W3C, 2004.
- [19] Teodor C. Przymusiński. On the declarative and procedural semantics of logic programs. *Journal of Automated Reasoning*, 5(2):167–205, 1989.
- [20] Nathalie Steinmetz and Ioan Toma (Eds.). WSML language reference. Working Draft D16.1 v1.0, WSML, 2008. <http://www.wsmo.org/TR/d16/d16.1/v1.0/>.