What exactly the propositions are depends on the specific logic under consideration. In our case, for instance, the propositions are RDF triples. In the appendix you can find another example of logic propositions in the context of first-order predicate logic. Given a specific logic, let us denote the set of all propositions by $\mathbb{P}$. Furthermore, we need a notation to state that, e.g., propositions p_3 and p_4 are logical consequences of the propositions p_1, and p_2. Most commonly, this is expressed by $\{p_1, p_2\} \models \{p_3, p_4\}$, where $\models$ is called *entailment relation* and relates sets of propositions with sets of propositions (hence: $\models \subseteq 2^{\mathbb{P}} \times 2^{\mathbb{P}}$). A logic L is therefore composed of a set of propositions together with an entailment relation and can be described by $L = (\mathbb{P}, \models)$ on an abstract level.

There are numerous ways to define the entailment relation of a specific logic. In the following, we will attend to a rather frequently employed method that is also used in the case of RDF(S).

3.2 Model-Theoretic Semantics for RDF(S)

We start by giving a high-level perspective of the notion of *model-theoretic semantics*. Thereby, one central notion is that of an *interpretation*. Interpretations might be conceived as potential "realities" or "worlds." In particular, interpretations need in no way comply with the actual reality. In formal logic, one usually chooses certain mathematical structures as interpretations in order to work in a formally correct way. Which structures to choose in particular depends on the considered logic.

After stipulating what the interpretations of a logic are, one proceeds by defining how to decide whether a specific interpretation I *satisfies* a specific proposition $p \in \mathbb{P}$ (in which case we call I *model* of p and write $I \models p$, using the same symbol as for the entailment relation). Moreover, for a set $P \subseteq \mathbb{P}$ of propositions, one says that I is a model of P (written $I \models P$), if it is a model for every $p \in P$.

Based on this "model relation" the actual entailment relation is defined in the following (also intuitively plausible) way: a proposition set $P' \subseteq \mathbb{P}$ is entailed by a set of propositions $P \subseteq \mathbb{P}$ (written: $P \models P'$) if and only if every interpretation I satisfying *all* sentences p from P (formally: $I \models P$) is also a model of every sentence p' from P' (i.e., $I \models P'$). Figure 3.1 depicts this correspondence graphically. To further illustrate the basic concept of this definition, consider the following (only halfway formal) analogy: "light green" entails "green," because all light green things are also just green. Using the terminology just introduced and thinking of interpretations as single real world objects, this can be expressed as follows: $\{light\_green\} \models green$, because every thing (every interpretation) I that satisfies *light_green* (i.e., $I \models light\_green$)

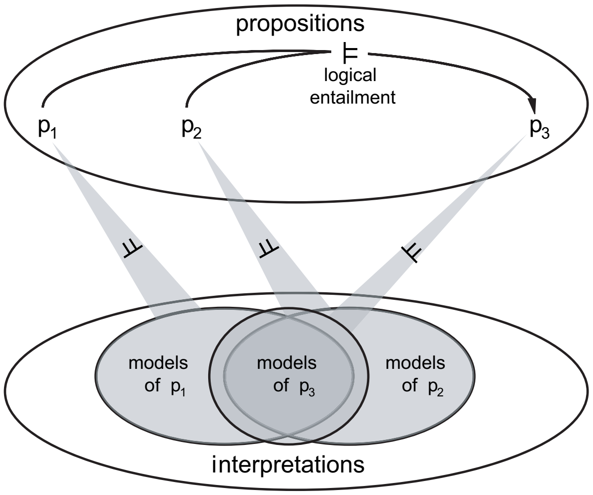

FIGURE 3.1: Definition of the entailment relation via models

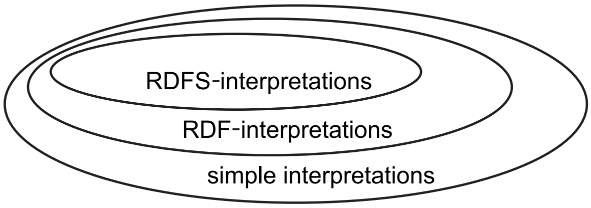

FIGURE 3.2: Correspondence of interpretations

is automatically also a model of the proposition *green* (i.e., $I \models green$).

We define the model-theoretic semantics for RDF(S) in several steps: we start by the comparably easy definition of simple interpretations of graphs. After that, we provide additional criteria which qualify these interpretations as RDF-interpretations. Finally we give further constraints to be fulfilled by an RDF-interpretation in order to be acknowledged as an RDFS-interpretation. As a natural consequence of this approach, every RDFS-interpretation is a valid RDF-interpretation and every RDF-interpretation constitutes a simple interpretation. This correspondency is depicted in Fig. 3.2.

3.2.1 Simple Interpretations

So, let us first have a look at the so-called *simple interpretations*. We shall use the Turtle syntax introduced in Section 2.2 in order to represent RDF graphs, presuming the two conventionally used prefix definitions

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
```

Our starting point for the definition of interpretations is the notion of vocabulary, introduced in Section 2.2.6 and further elaborated in Section 2.4. Formally, a vocabulary is just an arbitrary set containing URIs and literals.

Of course, the aim of the introduced semantics is to correctly reflect the intuition behind RDF graphs; hence the interpretations to be defined in the sequel should – although more abstract – in a certain sense be similar to the "possible worlds" resp. "realities" described by the graphs.

As pointed out in Chapter 2, triples are employed to describe how *resources* are interrelated via *properties*. Consequently, an interpretation contains two sets IR and IP, the elements of which can be understood as abstract resources resp. properties, as well as a function $\mathrm{I_{EXT}}$ that tells which resources are interconnected by which properties. So, "resource" and "property" are notions which are purely semantic and to be used on the interpretation side only, whence – in the strict sense – it would be wrong to say that URIs (or literals) *are* resources. More precisely, one should state that (syntactic) URIs or literals *stand for* or *represent* (semantic) resources. And exactly this kind of representation is encoded by further functions that assign a semantic counterpart to every URI and literal. In the case of simple interpretations, all URIs are treated equally as there is no "semantic special treatment" for the RDF and the RDFS vocabulary.

So we define: a *simple interpretation* $\mathcal{I}$ of a given vocabulary V consists of

- IR, a non-empty set of *resources*, alternatively called domain or universe of discourse of $\mathcal{I}$,
- IP, the set of *properties* of $\mathcal{I}$ (which may overlap with IR),
- $\mathrm{I_{EXT}}$, a function assigning to each property a set of pairs from IR, i.e. $\mathrm{I_{EXT}} : IP \to 2^{IR \times IR}$, where $\mathrm{I_{EXT}}(p)$ is called the *extension* of the property p,
- $\mathrm{I_S}$, a function, mapping URIs from V into the union of the sets IR and IP, i.e. $\mathrm{I_S} : V \to IR \cup IP$,
- $\mathrm{I_L}$, a function from the typed literals from V into the set IR of resources and
- LV, a particular subset of IR, called the set of *literal values*, containing (at least) all untyped literals from V.

Based on the sets IR, IP, and LV as well as the functions $\mathrm{I_{EXT}}$, $\mathrm{I_S}$, and $\mathrm{I_L}$, we now define an interpretation function $\cdot^{\mathcal{I}}$ that in the first place maps all literals and URIs contained in the vocabulary V to resources and properties:

- every untyped literal $"a"$ is mapped to a, formally: $("a")^{\mathcal{I}} = a$,
- every untyped literal carrying language information $"a"@t$ is mapped to the pair $\langle a, t\rangle$, i.e. $("a"@t)^{\mathcal{I}} = \langle a, t\rangle$,

- every typed literal l is mapped to $\mathrm{I_L}(l)$, formally: $l^{\mathcal{I}} = \mathrm{I_L}(l)$, and
- every URI u is mapped to $\mathrm{I_S}(u)$, i.e. $u^{\mathcal{I}} = \mathrm{I_S}(u)$.

Note that, as mentioned in Section 2.1.3, untyped literals without language information are essentially mapped to themselves, while untyped literals with language information are assigned to pairs consisting of the pure literal and the language identifier. Figure 3.3 graphically illustrates this part of the definition of a simple interpretation.

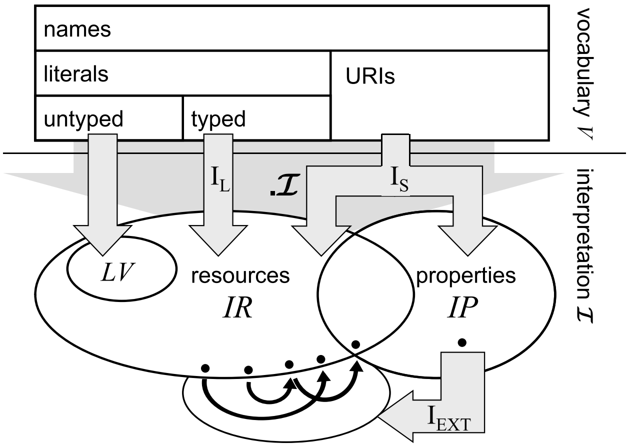

FIGURE 3.3: Schematic representation of a simple interpretation

Now, starting from the definition of the interpretation function with respect to single basic RDF elements, we further extend this function in a way that it assigns a truth value (*true* or *false*) to every *grounded* triple (i.e. every triple not containing blank nodes): the truth value `s p o.`$^{\mathcal{I}}$ of a grounded triple `s p o.` will be true exactly if all of its constituents `s`, `p`, and `o` are contained in the vocabulary V and additionally $\langle \mathtt{s}^{\mathcal{I}}, \mathtt{o}^{\mathcal{I}} \rangle \in \mathrm{I_{EXT}}(\mathtt{p}^{\mathcal{I}})$ holds. Verbally, the latter condition demands that the pair constructed from the resources assigned to `s` and `o` is within the extension of the property denoted by `p`. Figure 3.4 graphically displays this condition. If one of these mentioned conditions is violated, the truth value will be *false*.

Finally, the interpretation function $\cdot^{\mathcal{I}}$ also assigns a truth value to every grounded graph G: $G^{\mathcal{I}}$ is true if and only if every triple contained in the graph G is true, i.e. $G^{\mathcal{I}} = \mathit{true}$ exactly if $T^{\mathcal{I}} = \mathit{true}$ for all $T \in G$.

Mark that the notion of interpretation which we have introduced so far only covers grounded graphs, i.e. those not containing blank nodes. In order to enable an interpretation to deal with blank nodes, we have to further generalize our technical notion of interpretation. For this, the essential idea

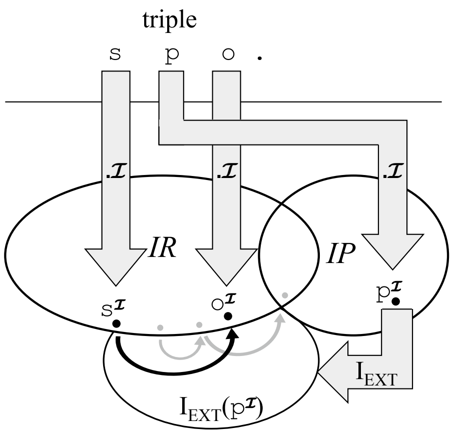

FIGURE 3.4: Criterion for the validity of a triple with respect to an interpretation

is to let a graph that contains blank nodes be valid, if everyone of these blank nodes can be replaced by a resource, such that the resulting bnode-free graph is valid. Hence, let A be a function assigning a resource from IR to every blank node occurring in G. Moreover, we define for such a mapping A and a given interpretation $\mathcal{I}$ a sort of combined interpretation $\mathcal{I}+A$ that behaves exactly like $\mathcal{I}$ on the URIs and literals but additionally uses A to assign resources to all blank nodes: $(b)^{\mathcal{I}+A} = A(b)$. Accordingly $\mathcal{I}+A$ can be extended to triples and further to graphs.

Eventually, we have to abstract from the concrete blank node assignments by stipulating that a (non-combined) interpretation $\mathcal{I}$ be a model of a graph G if there exists a function A', such that $G^{\mathcal{I}+A'} = \mathit{true}$. By this trick, we have extended our original notion of an interpretation to non-grounded graphs. An example of such a simple interpretation is given in Fig. 3.5.

In full compliance with the idea of model-theoretic semantics, we now say that a graph G_1 (simply) entails a graph G_2, if every simple interpretation that is a model of G_1 is also a model of G_2.

3.2.2 RDF-Interpretations

As mentioned earlier, simple interpretations essentially treat all URIs occurring in the vocabulary in the same way, irrespective of their namespace and their intended special meaning. For example, a simple interpretation does not semantically distinguish between the URIs `ex:publishedBy` and `rdf:type`. In order to restore the fixed vocabulary to its intended meaning, the set of admissible interpretations has to be further restricted by additional constraints.

The RDF vocabulary V_{RDF} consists of the URIs

Let us consider as an example the graph from Fig. 2.7. The corresponding vocabulary V consists of all names of nodes and edges of the graph.
A simple interpretation $\mathcal{I}$ for this vocabulary would now be given by:

$$\begin{aligned} IR &= \{\chi, \upsilon, \tau, \nu, \epsilon, \iota, 1lb\} \\ IP &= \{\tau, \nu, \iota\} \\ LV &= \{1lb\} \\ \mathrm{I_{EXT}} &= \tau \mapsto \{\langle \chi, \epsilon \rangle\} \\ & \quad\; \nu \mapsto \{\langle \epsilon, \upsilon \rangle\} \\ & \quad\; \iota \mapsto \{\langle \epsilon, 1lb \rangle\} \end{aligned}$$

$$\begin{aligned} \mathrm{I_S} = \; & \texttt{ex:chutney} \mapsto \chi \\ & \texttt{ex:greenMango} \mapsto \upsilon \\ & \texttt{ex:hasIngredient} \mapsto \tau \\ & \texttt{ex:ingredient} \mapsto \nu \\ & \texttt{ex:amount} \mapsto \iota \end{aligned}$$

$\mathrm{I_L}$ is the "empty function," since there are no typed literals.

Letting A : `_:id1` $\mapsto \epsilon$, we note that the interpretation $\mathcal{I}+A$ valuates all three triples of our considered graph with *true*:

$$\begin{aligned} &\langle \texttt{ex:chutney}^{\mathcal{I}+A}, \texttt{\_:id1}^{\mathcal{I}+A} \rangle = \langle \chi, \epsilon \rangle \in \mathrm{I_{EXT}}(\tau) = \mathrm{I_{EXT}}(\texttt{ex:hasIngredient}^{\mathcal{I}+A}) \\ &\langle \texttt{\_:id1}^{\mathcal{I}+A}, \texttt{ex:greenMango}^{\mathcal{I}+A} \rangle = \langle \epsilon, \upsilon \rangle \in \mathrm{I_{EXT}}(\nu) = \mathrm{I_{EXT}}(\texttt{ex:ingredient}^{\mathcal{I}+A}) \\ &\langle \texttt{\_:id1}^{\mathcal{I}+A}, \texttt{"1lb"}^{\mathcal{I}+A} \rangle = \langle \epsilon, 1lb \rangle \in \mathrm{I_{EXT}}(\iota) = \mathrm{I_{EXT}}(\texttt{ex:amount}^{\mathcal{I}+A}) \end{aligned}$$

Therefore, the described graph as a whole is also valued with *true*. Hence, the simple interpretation $\mathcal{I}$ is a model of the graph.

FIGURE 3.5: Example of an interpretation

`rdf:type rdf:Property rdf:XMLLiteral rdf:nil rdf:List rdf:Statement`
`rdf:subject rdf:predicate rdf:object rdf:first rdf:rest rdf:Seq`
`rdf:Bag rdf:Alt rdf:value`

plus an infinite number of URIs `rdf_`i for every positive integer i.

Recall the intuitive semantics for this vocabulary: `rdf:type` is used to assign a type to a URI; in other words, it declares that the resource associated to this URI belongs to a certain class. The name `rdf:Property` denotes such a class and characterizes all those URIs that may serve as a triple's predicate, i.e. those URIs whose assigned resources have an extension (i.e. which are in *IP* in terms of simple interpretations). Consequently, only interpretations satisfying those conditions will be admitted.

As we learned in Section 2.3.1, there is exactly one predefined datatype in RDF, namely, `rdf:XMLLiteral`. As opposed to other (externally defined) datatypes, the special characteristics of this one are explicitly taken care of in the RDF semantics definition. In order to do this, it is necessary to distinguish between *well-typed* and *ill-typed* XML literals. An XML literal is categorized as well-typed if it satisfies the syntactic conditions for being contained in the lexical space of `rdf:XMLLiteral`; otherwise it is ill-typed.

This distinction is relevant for the subsequent definition, because well-typed literals are mapped to literal values (i.e. elements of *LV*), whereas ill-typed ones are mapped to resources that are not literal values.

An *RDF-interpretation* of a vocabulary V is a simple interpretation of the vocabulary $V \cup V_{\mathrm{RDF}}$ that additionally satisfies the following conditions:

- $x \in IP$ exactly if $\langle x, \texttt{rdf:Property}^{\mathcal{I}}\rangle \in \mathrm{I}_{\mathrm{EXT}}(\texttt{rdf:type}^{\mathcal{I}})$.

  x is a property exactly if it is connected to the resource denoted by `rdf:Property` via the `rdf:type`-property (this automatically causes $IP \subseteq IR$ for any RDF-interpretation).

- ~~if `"`s`"^^rdf:XMLLiteral` is contained in V and s is a well-typed XML-Literal, then~~

  - ~~$\mathrm{I_L}(\texttt{"}s\texttt{"\^{}\^{}rdf:XMLLiteral})$ is the XML value[1] of s;~~
  - ~~$\mathrm{I_L}(\texttt{"}s\texttt{"\^{}\^{}rdf:XMLLiteral}) \in LV$;~~
  - ~~$\langle \mathrm{I_L}(\texttt{"}s\texttt{"\^{}\^{}rdf:XMLLiteral}), \texttt{rdf:XMLLiteral}^{\mathcal{I}}\rangle \in \mathrm{I}_{\mathrm{EXT}}(\texttt{rdf:type}^{\mathcal{I}})$~~

[1]The value space of the datatype assigned to `rdf:XMLLiteral` contains, for every well-typed XML string (from the lexical space), exactly one so-called XML value. The RDF specification does not give further information about the nature of XML values; it only requires that an XML value is not an XML string, nor a data value, nor a Unicode string. For our purposes and the intuitive usage, however, it does no harm to suppose that XML values are just XML strings.

- if `"`*s*`"^^rdf:XMLLiteral` is contained in V and *s* is an ill-typed XML literal, then
  - $\mathrm{I_L}($`"`*s*`"^^rdf:XMLLiteral`$) \notin LV$ and
  - $\langle \mathrm{I_L}($`"`*s*`"^^rdf:XMLLiteral`$),$ `rdf:XMLLiteral`$^{\mathcal{I}}\rangle \notin \mathrm{I_{EXT}}($`rdf:type`$^{\mathcal{I}})$.

In addition to those semantic restrictions, RDF-interpretations have to satisfy the condition that all of the subsequent triples (called *axiomatic triples*) must be valued as true:

`rdf:type`	`rdf:type`	`rdf:Property`.
`rdf:subject`	`rdf:type`	`rdf:Property`.
`rdf:predicate`	`rdf:type`	`rdf:Property`.
`rdf:object`	`rdf:type`	`rdf:Property`.
`rdf:first`	`rdf:type`	`rdf:Property`.
`rdf:rest`	`rdf:type`	`rdf:Property`.
`rdf:value`	`rdf:type`	`rdf:Property`.
`rdf:_`i	`rdf:type`	`rdf:Property`.
`rdf:nil`	`rdf:type`	`rdf:List`.

Again, the i in `rdf:_`i is to be replaced by all positive integers; therefore we actually have infinitely many axiomatic triples.

Except for the last one, all those triples serve the purpose of marking resources that are assigned to particular RDF URIs as properties. This is done in the usual way by typing them with `rdf:type rdf:Property` which due to the above definition of RDF-interpretations has exactly the desired effect.

Together, the listed restrictions ensure that an RDF-interpretation complies with the intended meaning.

In exact analogy to the definition of the simple entailment, we now say that a graph G_1 RDF-entails a graph G_2 if every RDF-interpretation that is a model of G_1 is also a model of G_2.

3.2.3 RDFS Interpretations

As pointed out in Section 2.4, RDFS enriches the RDF vocabulary by further constructs which have to be interpreted in a special way. For example, new class names are introduced that allow us to mark a URI as referring to a resource, to an untyped literal, or to a class via `rdf:type`. New URIs for properties allow for characterizing domain and range of a property by typing them with classes. Moreover class names as well as property names can be put into hierarchical relations. This set of modeling options enables us to express schematic or terminological knowledge in the form of triples.

The RDFS vocabulary V_{RDFS} to be specifically interpreted consists of the following names:

`rdfs:domain rdfs:range rdfs:Resource rdfs:Literal rdfs:Datatype`
`rdfs:Class rdfs:subClassOf rdfs:subPropertyOf rdfs:member`
`rdfs:Container rdfs:ContainerMembershipProperty rdfs:comment`
`rdfs:seeAlso rdfs:isDefinedBy rdfs:label`

For the sake of a simpler presentation, we introduce a new function $\mathrm{I}_{\mathrm{CEXT}}$ which, given a fixed RDF-interpretation, maps resources to sets of resources (formally: $\mathrm{I}_{\mathrm{CEXT}} : IR \to 2^{IR}$). We define $\mathrm{I}_{\mathrm{CEXT}}(y)$ to contain exactly those elements x for which $\langle x, y\rangle$ is contained in $\mathrm{I}_{\mathrm{EXT}}(\texttt{rdf:type}^{\mathcal{I}})$. The set $\mathrm{I}_{\mathrm{CEXT}}(y)$ is then also called the *(class) extension* of y.

Moreover we let IC denote the class extension of the URI `rdfs:Class`, formally: $IC = \mathrm{I}_{\mathrm{CEXT}}(\texttt{rdfs:Class}^{\mathcal{I}})$. Note that both $\mathrm{I}_{\mathrm{CEXT}}$ as well as IC are uniquely determined by $\cdot^{\mathcal{I}}$ and $\mathrm{I}_{\mathrm{EXT}}$.

We now employ the newly introduced function in order to specify the semantic requirements on an RDFS-interpretation:

An *RDFS-interpretation* of a vocabulary V is an RDF-interpretation of the vocabulary $V \cup V_{\mathrm{RDFS}}$ that in addition satisfies the following criteria:

- $IR = \mathrm{I}_{\mathrm{CEXT}}(\texttt{rdfs:Resource}^{\mathcal{I}})$

  Every resource has the type `rdfs:Resource`.

- $LV = \mathrm{I}_{\mathrm{CEXT}}(\texttt{rdfs:Literal}^{\mathcal{I}})$

  Every untyped or well-typed literal has the type `rdfs:Literal`.

- If $\langle x, y\rangle \in \mathrm{I}_{\mathrm{EXT}}(\texttt{rdfs:domain}^{\mathcal{I}})$ and $\langle u, v\rangle \in \mathrm{I}_{\mathrm{EXT}}(x)$,
  then $u \in \mathrm{I}_{\mathrm{CEXT}}(y)$.

  If x and y are interconnected by the property `rdfs:domain` and the property x connects the resources u and v, then u has the type y.

- If $\langle x, y\rangle \in \mathrm{I}_{\mathrm{EXT}}(\texttt{rdfs:range}^{\mathcal{I}})$ and $\langle u, v\rangle \in \mathrm{I}_{\mathrm{EXT}}(x)$,
  then $v \in \mathrm{I}_{\mathrm{CEXT}}(y)$.

  If x and y are interconnected by the property `rdfs:range` and the property x connects the resources u and v, then v has the type y.

- $\mathrm{I}_{\mathrm{EXT}}(\texttt{rdfs:subPropertyOf}^{\mathcal{I}})$ is reflexive and transitive on IP.

  The `rdfs:subPropertyOf` property connects every property with itself.

  Moreover: if `rdfs:subPropertyOf` links property x with property y and also y with the property z, then `rdfs:subPropertyOf` also links x directly with z.

- If $\langle x, y\rangle \in \mathrm{I}_{\mathrm{EXT}}(\texttt{rdfs:subPropertyOf}^{\mathcal{I}})$,
  then $x, y \in IP$ and $\mathrm{I}_{\mathrm{EXT}}(x) \subseteq \mathrm{I}_{\mathrm{EXT}}(y)$.

  Whenever x and y are interlinked by `rdfs:subPropertyOf`, then both x and y are properties and every pair of resources contained in x's extension is also contained in the extension of y.

- If $x \in IC$,
  then $\langle x, \texttt{rdfs:Resource}^{\mathcal{I}}\rangle \in \mathrm{I}_{\mathrm{EXT}}(\texttt{rdfs:subClassOf}^{\mathcal{I}})$.

  Every class x is a subclass of the class of all resources, i.e. the pair constructed from x and `rdfs:Resource` is in the extension of `rdfs:subClassOf`.

- If $\langle x, y\rangle \in \mathrm{I}_{\mathrm{EXT}}(\texttt{rdfs:subClassOf}^{\mathcal{I}})$,
  then $x, y \in IC$ and $\mathrm{I}_{\mathrm{CEXT}}(x) \subseteq \mathrm{I}_{\mathrm{CEXT}}(y)$.

  If x and y are in the `rdfs:subClassOf` relation, then both x and y are classes and the (class) extension of x is a subset of the (class) extension of y.

- $\mathrm{I}_{\mathrm{EXT}}(\texttt{rdfs:subClassOf}^{\mathcal{I}})$ is reflexive and transitive on IC.

  The `rdfs:subClassOf` property connects each class with itself.

  Moreover if the `rdfs:subClassOf` property connects a class x with a class y and y with some class z, it also connects x with z directly.

- ~~If $x \in \mathrm{I}_{\mathrm{CEXT}}(\texttt{rdfs:ContainerMembershipProperty}^{\mathcal{I}})$,
  then $\langle x, \texttt{rdfs:member}^{\mathcal{I}}\rangle \in \mathrm{I}_{\mathrm{EXT}}(\texttt{rdfs:subPropertyOf}^{\mathcal{I}})$.~~

  ~~Any property typed with `rdfs:ContainerMembershipProperty` is in the `rdfs:subPropertyOf` relation to the `rdfs:member` property.~~

- ~~If $x \in \mathrm{I}_{\mathrm{CEXT}}(\texttt{rdfs:Datatype}^{\mathcal{I}})$,
  then $\langle x, \texttt{rdfs:Literal}^{\mathcal{I}}\rangle \in \mathrm{I}_{\mathrm{EXT}}(\texttt{rdfs:subClassOf}^{\mathcal{I}})$~~

  ~~Any x typed as `rdfs:Datatype` must be a subclass of the class of all literal values (denoted by `rdfs:Literal`).~~

~~In analogy to the definition of RDF-interpretations, we name a list of axiomatic triples which (in addition to the aforementioned constraints) have to be satisfied by an RDF-interpretation in order to render it an RDFS-interpretation:~~

~~`rdf:type`~~	~~`rdfs:domain`~~	~~`rdfs:Resource .`~~
~~`rdfs:domain`~~	~~`rdfs:domain`~~	~~`rdf:Property .`~~
~~`rdfs:range`~~	~~`rdfs:domain`~~	~~`rdf:Property .`~~
~~`rdfs:subPropertyOf`~~	~~`rdfs:domain`~~	~~`rdf:Property .`~~
~~`rdfs:subClassOf`~~	~~`rdfs:domain`~~	~~`rdfs:Class .`~~
~~`rdf:subject`~~	~~`rdfs:domain`~~	~~`rdf:Statement .`~~
~~`rdf:predicate`~~	~~`rdfs:domain`~~	~~`rdf:Statement .`~~
~~`rdf:object`~~	~~`rdfs:domain`~~	~~`rdf:Statement .`~~
~~`rdfs:member`~~	~~`rdfs:domain`~~	~~`rdfs:Resource .`~~
~~`rdf:first`~~	~~`rdfs:domain`~~	~~`rdf:List .`~~
~~`rdf:rest`~~	~~`rdfs:domain`~~	~~`rdf:List .`~~
~~`rdfs:seeAlso`~~	~~`rdfs:domain`~~	~~`rdfs:Resource .`~~
~~`rdfs:isDefinedBy`~~	~~`rdfs:domain`~~	~~`rdfs:Resource .`~~

```
rdfs:comment        rdfs:domain         rdfs:Resource .
rdfs:label          rdfs:domain         rdfs:Resource .
rdf:value           rdfs:domain         rdfs:Resource .

rdf:type            rdfs:range          rdfs:Class .
rdfs:domain         rdfs:range          rdfs:Class .
rdfs:range          rdfs:range          rdfs:Class .
rdfs:subPropertyOf  rdfs:range          rdf:Property .
rdfs:subClassOf     rdfs:range          rdfs:Class .
rdf:subject         rdfs:range          rdfs:Resource .
rdf:predicate       rdfs:range          rdfs:Resource .
rdf:object          rdfs:range          rdfs:Resource .
rdfs:member         rdfs:range          rdfs:Resource .
rdf:first           rdfs:range          rdfs:Resource .
rdf:rest            rdfs:range          rdf:List .
rdfs:seeAlso        rdfs:range          rdfs:Resource .
rdfs:isDefinedBy    rdfs:range          rdfs:Resource .
rdfs:comment        rdfs:range          rdfs:Literal .
rdfs:label          rdfs:range          rdfs:Literal .
rdf:value           rdfs:range          rdfs:Resource .

rdfs:ContainerMembershipProperty
                    rdfs:subClassOf     rdf:Property .
rdf:Alt             rdfs:subClassOf     rdfs:Container .
rdf:Bag             rdfs:subClassOf     rdfs:Container .
rdf:Seq             rdfs:subClassOf     rdfs:Container .

rdfs:isDefinedBy    rdfs:subPropertyOf  rdfs:seeAlso .

rdf:XMLLiteral      rdf:type            rdfs:Datatype .
rdf:XMLLiteral      rdfs:subClassOf     rdfs:Literal .
rdfs:Datatype       rdfs:subClassOf     rdfs:Class .

rdf:_i              rdf:type
                      rdfs:ContainerMembershipProperty .
rdf:_i              rdfs:domain         rdfs:Resource .
rdf:_i              rdfs:range          rdfs:Resource .
```

Again, i can be replaced by any positive integer. Obviously this set of triples can be divided into several groups. The first group contains triples with predicate `rdfs:domain`. The declarative purpose of such a triple `p rdfs:domain c` is to associate the URI `p` with a class name `c`. Basically, this enforces a class membership (realized via `rdf:type`) for every URI `s` occurring as a subject together with the predicate `p` in a triple `s p o` . For example, the fifth triple in this list just states that whenever a triple `c rdfs:subclassOf d` is encountered, an immediate consequence is that `c` denotes a class, expressed by the triple `c rdf:type rdfs:Class`.

Similarly, the triples gathered in the second group and having the predicate `rdfs:range` cause class memberships of triple objects.

As to containers, the axiomatic triples specify the class of all containedness properties as subclass of the class of all properties. Additionally, the class denoted by `rdfs:Container` is declared as the superclass of all kinds of open lists.

Moreover, the `rdfs:isDefinedBy` property is classified as a special case of the `rdfs:seeAlso` property. The class of XML values is marked as a datatype and subclass of all literal values, and the class of all datatypes is identified as a class of classes.

Finally the predefined containedness properties for lists are characterized as such.

Based on the introduced notion of an RDFS-interpretation and in analogy to the previous two cases, we now define that a graph G_1 RDFS entails a graph G_2 if every RDFS-interpretation that is a model of G_1 is also a model of G_2.

3.2.4 Interpretation of Datatypes

We already know that there is just one predefined datatype in RDFS, namely, `rdf:XMLLiteral`, the semantic characteristics of which are fully covered by the definition of RDFS-interpretation in the previous section. Nevertheless, other externally defined datatypes can be used in RDF(S).

In Section 2.3.1 we learned that a datatype d is composed of a *value space* Val_d, a *lexical space* Lex_d and a function $\mathrm{Lex2Val}_d$, assigning a value to every element of the lexical space, formally: $d = \langle Val_d, Lex_d, \mathrm{Lex2Val}_d\rangle$ with $\mathrm{Lex2Val}_d : Lex_d \to Val_d$.

In the same section we also mentioned that when employing external datatypes, one can have URIs referring to those datatypes within the vocabulary. This allows for making statements about datatypes within an RDF(S) specification. For example, it might be reasonable to specify that the natural numbers are a subset of the integers.

In order to capture the entirety of all datatypes used in an RDF(S) description, we introduce the notion of a *datatype map* D, a function assigning the datatypes to their URIs: $D : \mathtt{u} \mapsto d$. Of course, the predefined datatype has to be treated accordingly; hence we require every datatype map to satisfy $D(\mathtt{rdf:XMLLiteral}) = d_{\mathrm{XMLLiteral}}$.

Given a datatype map D, we now define a *D-interpretation* of the vocabulary V as an RDFS-interpretation $\mathcal{I}$ of $V \cup \{\mathtt{a} \mid \text{ there is a } d \text{ with } D(\mathtt{a}) = d\}$ (that means the vocabulary V extended by the domain of D) that for every $\mathtt{a}$ and d with $D(\mathtt{a}) = d$ additionally satisfies the following properties:

- $\mathtt{a}^{\mathcal{I}} = D(\mathtt{a})$.

~~For URIs denoting datatypes, the interpretation function $\cdot^{\mathcal{I}}$ coincides with the datatype map D.~~

- ~~$\mathrm{I}_{\mathrm{CEXT}}(d) = \mathit{Val}_d \subseteq \mathit{LV}$.~~

  ~~The class extension of a datatype d is the value space of d and is a subset of the literal values.~~

- ~~For every typed literal `"s"^^d` $\in V$ with `d`$^{\mathcal{I}} = d$ the following hold:~~

  - ~~if $\mathtt{s} \in \mathit{Lex}_d$, then $\mathrm{I}_{\mathrm{L}}($`"s"^^d`$) = \mathrm{Lex2Val}_d(\mathtt{s})$,~~
  - ~~if $\mathtt{s} \notin \mathit{Lex}_d$, then $\mathrm{I}_{\mathrm{L}}($`"s"^^d`$) \notin \mathit{LV}$.~~

  ~~Every well-typed literal (i.e. one contained in the lexical space of its associated datatype) is mapped into the literal values in accordance with this datatype's lexical-to-value mapping, whereas every ill-typed literal is mapped to a resource outside the literal values.~~

- ~~$\mathtt{a}^{\mathcal{I}} \in \mathrm{I}_{\mathrm{CEXT}}($`rdfs:Datatype`$^{\mathcal{I}})$.~~

  ~~Every datatype (i.e. every resource assigned to a datatype URI) is a member of the `rdfs:Datatype` class.~~

3.2.5 Worked Example

Let us have a closer look at the definitions of models for RDF and RDFS documents by working through them for the example ontology from Section 2.6. This is going to be a bit tedious, but it helps to understand the definitions. Usually, you would not do this manually, but rather use systems based on algorithms like that from Section 3.3.

Let us start by defining a simple interpretation, as given in Fig. 3.6. These assignments define a simple interpretation which is a model of the example ontology. You can check this easily yourself.

Next, we define an RDF-interpretation starting from the simple interpretation just given. To do this, we need to augment the simple interpretation by adding mappings for all elements of V_{RDF} and by redefining $\mathrm{I}_{\mathrm{EXT}}(y)$.

It does not really matter how we set $\mathrm{I}_{\mathrm{S}}(x)$ for those $x \in V_{\mathrm{RDF}}$ which the I_{S} from the simple interpretation does not map, so pick anything that is not yet in $\mathit{IR} \cup \mathit{IP}$ and extend I_{S} accordingly. Note that we could also reuse the elements from $\mathit{IR} \cup \mathit{IP}$ because the unique name assumption is not imposed, but we want to construct a model which is intuitively feasible, and so we avoid reuse. Let's do the settings as given in Fig. 3.7.

Now redefine $\mathrm{I}_{\mathrm{EXT}}(y)$ from the simple interpretation to the following:

$$\mathrm{I}_{\mathrm{EXT}}(y) = \{\langle s,a\rangle, \langle h,i\rangle, \langle d,\pi\rangle, \langle e,\pi\rangle, \langle h,\pi\rangle, \langle b,\pi\rangle, \langle m,\pi\rangle, \langle o,\pi\rangle, \langle r,\pi\rangle, \langle y,\pi\rangle,$$
$$\langle \rho_1,\rho_2\rangle, \langle \rho_4,\pi\rangle, \langle \rho_5,\pi\rangle, \langle \rho_6,\pi\rangle, \langle \rho_7,\pi\rangle, \langle \rho_8,\pi\rangle, \langle \rho_{12},\pi\rangle, \langle \delta_k,\pi\rangle \mid k \in \mathbb{N}\}.$$

This way, we satisfy the first condition on page 80. The other conditions are not important for us since we have no such elements in V.

$$
\begin{aligned}
IR &= \{a, c, i, n, p, s, t, v, y, d, h\}\\
IP &= \{d, e, h, b, m, o, r, y\}\\
LV &= \emptyset\\
\mathrm{I_S} &= \texttt{ex:AllergicToNuts} && \mapsto a\\
& \texttt{ex:coconutMilk} && \mapsto c\\
& \texttt{ex:Nutty} && \mapsto n\\
& \texttt{ex:Pitiable} && \mapsto p\\
& \texttt{ex:sebastian} && \mapsto s\\
& \texttt{ex:Thai} && \mapsto t\\
& \texttt{ex:vegetableThaiCurry} && \mapsto v\\
& \texttt{ex:thaiDishBasedOn} && \mapsto d\\
& \texttt{ex:eats} && \mapsto e\\
& \texttt{ex:hasIngredient} && \mapsto h\\
& \texttt{rdfs:subPropertyOf} && \mapsto b\\
& \texttt{rdfs:ContainerMembershipProperty} && \mapsto i\\
& \texttt{rdfs:domain} && \mapsto m\\
& \texttt{rdfs:subClassOf} && \mapsto o\\
& \texttt{rdfs:range} && \mapsto r\\
& \texttt{rdf:type} && \mapsto y\\
\mathrm{I_{EXT}} &= d \mapsto \{\langle v, c\rangle\}\\
& e \mapsto \{\langle s, v\rangle\}\\
& h \mapsto \emptyset\\
& b \mapsto \{\langle d, h\rangle\}\\
& m \mapsto \{\langle d, t\rangle\}\\
& o \mapsto \{\langle a, p\rangle\}\\
& r \mapsto \{\langle d, n\rangle\}\\
& y \mapsto \{\langle s, a\rangle, \langle h, i\rangle\}\\
\mathrm{I_L} &= \emptyset
\end{aligned}
$$

FIGURE 3.6: Example of a simple interpretation