Analyzing Schema.org

Peter F. Patel-Schneider

Nuance Communications
pfpschneider@gmail.com

Abstract. Schema.org is a way to add machine-understandable information to web pages that is processed by the major search engines to improve search performance. The definition of schema.org is provided as a set of web pages plus a partial mapping into RDF triples with unusual properties, and is incomplete in a number of places. This analysis of and formal semantics for schema.org provides a complete basis for a plausible version of what schema.org should be.

Schema.org[1] "provides a collection of schemas, i.e., html tags, that webmasters can use to [mark up] their pages in ways recognized by major search providers."[2] The major search engine providers, including Bing, Google, Yahoo!, and Yandex use schema.org markup to improve the display of search results and schema.org has been designed by and is controlled by these organizations. This makes schema.org markup an important kind of machine-understandable data in the web. Not only are there many web pages with schema.org information, but this information is used in important ways.

Aside from being a collection of schemas, schema.org is a language for representing information on the Web, different from other languages used for this purpose, such as RDF [1, 2], OWL [3, 4], and the language underlying Freebase [5]. Using this language, the schema.org schemas are organized into a simple taxonomy by generalization relationships and other ontolological aspects of schema.org information are specified.

The publicly available definition of schema.org is, however, incomplete and contradictory. It is only provided as English text on various web pages in *schema.org*, plus mappings of the collection of schemas[3] into RDF (*http://schema.org/docs/full_md.html*) and OWL (*http://schema.org/docs/schemaorg.owl*). The RDF mapping centrally uses non-RDFS properties, such as *http://schema.org/domainIncludes*, so it is not possible to determine the meaning of schema.org constructs from the RDF mapping. The OWL mapping is somewhat better, as domains and ranges employ OWL unions, but the mapping is only a translation of part of what defines schema.org. The lack of a complete definition of schema.org limits the possibility of extracting the correct information from web pages that have schema.org markup.

This paper provides a full basis for schema.org as it should be, filling in the holes in the available descriptions of schema.org and fixing up discrepancies. The paper provides both a pre-theoretic analysis of schema.org and an abstract syntax and formal model-theoretic semantics for schema.org. This paper does not, however, draw on

[1] Throughout this paper schema.org refers to the general idea and *schema.org* refers to the collection of documents available at the *https://schema.org* web site.

[2] From *https://schema.org*, as of 1 April 2014.

[3] See *http://schema.org/docs/datamodel.html*.

the use of schema.org on web pages. Researchers can use the basis provided here to further investigate the properties of schema.org and schema.org markup. Providers of schema.org data can use this basis to reliably determine the meaning of the schema.org data they create. Developers can use this basis to build software that uses schema.org markup as information in a way that is compatible with the description of schema.org.

Description of Schema.org at *schema.org*

The description of schema.org in this section of the paper is taken from information on the web pages in *schema.org*, as of 1 April 2014. It ignores most of the surface syntax aspects of schema.org, concentrating on the underlying concepts and their intent.

Schema.org information is about items, e.g., the movie Avatar. Items can have types, e.g., the type identified by the URL *http://schema.org/Movie*. Items can have associated property-value pairs, e.g., the property identified by *http://schema.org/director* with value *"James Cameron"*. The value in a property value pair can be text, i.e., a Unicode string; a literal, e.g., a number or date; a URL, which identifies an item; or another item. There is no requirement that properties have only a single value for an item. Items can have associated URLs, e.g., *http://www.avatarmovie.com/index.html* and *http://en.wikipedia.org/wiki/Avatar_(2009_film)*, each of which identifies the item.

Schema.org provides a collection of types, via pages in *schema.org*, organized in a multi-parent generalization hierarchy. Each type is identified by the URL of the page that provides its definition. Each type has a set of parents, i.e., more-general types. Each type, except for datatypes, has a set of allowable properties for the type.

The types that are more specific than *http://schema.org/Enumeration* are enumeration types that also specify a set of URLs identifying all the items that are instances of the type. Datatypes are the types more specific than *http://schema.org/Datatype* and implicitly provide a set of non-item data values for them and a mapping from text to these values.

Schema.org also provides a collection of properties, again from *schema.org*, which may be also organized in a multi-parent generalization hierarchy.[4] Each property is identified by the URL of the page that provides its definition. Each property may have one or more types as domains, and can be used on items belonging to any of these types. Each property has one or more types as ranges, and values for the property belong to one or more of these types. However, property values can always be provided as just text.

There is a description of an extension mechanisms for schema.org, which only permits very simple extensions. It appears that the extension mechanism exists only to further subdivide existing schema.org properties, classes, and enumeration items and that these extensions are ignored within schema.org.

The translations of the type and property definitions of schema.org into RDF and OWL abide by the above description, except that there is no translation for the property hierarchy. These translations provide no extra information beyond what is given here.

[4] At the time of writing of this paper there was no general notation of the property hierarchy. While this paper was in review, the property hierarchy was officially announced (*http://lists.w3.org/Archives/Public/public-vocabs/2014Jun/0095.html*).

Analysis of *schema.org* as a Description of Schema.org

There are quite a number of aspects of schema.org and schema.org markup that are left unspecified in *schema.org*, are unclear, or raise issues. This section describes these aspects and provides extra assumptions that will be used in the account for schema.org presented here. The extra assumptions have been made in a way that is congruent with the information on *schema.org*, that make sense in an environment where there are large central consumers of large amounts of data, and that generate a reasonable representation formalism. (In several places, the comments in *schema.org* do not match the actual class or property, for example, instances of *http://schema.org/StructuredValue* are not strings, but this sort of mismatch is not the subject of this paper.)

It is unclear whether types and properties can also be items. However, items work quite differently from types and properties, and having arbitrary web pages being able to modify the types and properties of schema.org leads to difficulties, such as not being able to determine when a property is valid for an item until after all item information has been processed, so this account treats types and properties as being different from items. In particular, in this account different URLs that identify the same item do not identify the same type or the same property. Data values also act differently from items, so this account treats them as being disjoint from types, properties, and items. The identifiers of types and properties are different in schema.org, as URLs for types have initial capitals and URLs for properties do not, so it is fairly obvious that types are disjoint from properties.

Schema.org uses URLs as identifiers. URLs can be used to retrieve web pages, and this aspect of URLs is a main basis of schema.org. URLs officially can include fragment ids, and such URLs then identify parts of web pages. Although fragment identifiers are not currently used for any types and properties in *schema.org*, there is nothing technical preventing their use, and so they will be allowed in the account herein for types, properties, and items.

It is unclear whether schema.org types and properties must be identified by URLs in *schema.org*, but all current schema.org types and properties are so identified. This account does not formally make the assumption that types and properties must be identified by URLs in *schema.org*, but some of the pragmatic analysis does make the assumption that type and property definitions change infrequently, as is the case for types and properties identified by URLs in *schema.org*.

The mechanisms for working with datatypes are underspecified in *schema.org*. This account adds in a formal mechanism for determining the set of values for a datatype and a formal method for determining the data value corresponding to a text string for the datatype.

The name of the most general datatype in schema.org is *http://schema.org/Datatype*. This is an unfortunate name—*http://schema.org/Literal* would be much better—but the schema.org name will be used in this account. The name of the datatype for floating point numbers in schema.org is *http://schema.org/Float*. *http://schema.org/Float* and *http://schema.org/Integer* both have generalization *http://schema.org/Number*. This can lead to problems because floating point numbers are imprecise whereas integers are precise. This account, however, does not address the issue.

It is unclear whether the instances of an enumeration have to be items, or can also be data values. This account assumes that the instances of an enumeration are given as URLs, as is the case for all examples currently in schema.org, and thus that instances of an enumeration are items, not data values.

Some examples in *schema.org* only make sense if different but similar URLs identify different items. This is particularly the case for URLs that make up enumerations. This account assumes that different URLs in an enumeration identify different items, but does not otherwise assume that different URLs in the same namespace, e.g., different Wikipedia URLs, or in the same document identify different items. This extra assumption would be easy to add.

The domains of a property are specified both as part of types and as part of properties in *schema.org*. In all the examples there is no divergence between the two specifications, but the possibility of divergence is not ruled out. This account treats the specification in the type as the actual specification, as that seems to make more sense for disjunctive domains.

Because several properties indicate that they are subproperties of other properties, this account incorporates a multi-parent property hierarchy. There are some additions to the account herein that have to be made to support the property hierarchy.

Both domains and ranges of properties are disjunctive. This is different from most other representation formalisms, such as description logics [6] and RDF [2]. The stated rationale for this decision is that it reduces the need for general types that exist only to be domains or ranges. However, disjunctive domains and ranges mean that additions to a collection of schema.org information can be non-monotonic. The disjunctive nature of domains and ranges is fully explored in this account, including how it interacts with the property hierarchy.

Several aspects of the predominant syntaxes for schema.org markup obscure the workings of schema.org. This account transforms these aspects of surface syntax into a different abstract syntax.

Several types and properties are used as part of the foundations of schema.org in *schema.org*. Nearly all uses of these types and properties as general types or properties undermines the foundations of schema.org, so their use is disallowed in this account. The extension mechanism for schema.org is of very limited utility and appears to not have any effect on the processing of schema.org markup, so it is ignored in this account.

Description of Schema.org as It Should Be

This section contains a pre-theoretic description of schema.org and schema.org content as it should be, consonant with the discussion in the previous section. This description is designed to say how schema.org could work in a way that can be easily turned into a formal definition of schema.org, as is done in the following section of this paper.

Throughout this account, a URL is a uniform resource locator, optionally including a fragment part. The document (fragment) at that URL is (the appropriate fragment of) the document obtained by the usual web mechanisms for retrieving a document given a URL. URLs will be generally written as CURIES [7], with the prefixes *s* expanding to *http://schema.org/* and *w* expanding to *http://en.wikipedia.org/wiki/*, and the prefixes *rdf*, *rdfs*, and *owl* expanding to their usual expansions. The constituents of schema.org information are types, properties, data values, and items.

There is a collection of types, in a multi-parent generalization taxonomy, with two roots, *s:Thing* and *s:Datatype*. Each type is identified by a unique URL. The document (fragment) at that URL defines the type, listing:

1. some types that are more general than it (its parents), and
2. for non-datatypes, its properties (see below).

Parents and properties, and information about instances where appropriate, are the only information about a type obtainable from its defining document (fragment).

Each type has as a generalization (not necessarily directly specified in its defining document) either *s:Thing* or *s:Datatype*, but not both.

The types with strict generalization *s:Datatype* are datatypes. All the data values belonging to the datatype are described in the datatype's defining document (fragment), as is a way of transforming text strings into these data values. The datatypes are *s:Boolean*, *s:Date*, *s:DateTime*, *s:Number*, *s:Float*, *s:Integer*, *s:Text* (Unicode strings), *s:URL*, and *s:Time*. The details of these datatypes do not matter for this account, except for *s:Text*, and are not described here.

The type *s:Enumeration* has *s:Thing* as a parent.[5] Those types with strict generalization *s:Enumeration* are enumeration types. All those items with the enumeration type as a direct type are listed in the type's defining document (fragment). Different URLs identify different items in an enumeration.

The type *s:Thing* has properties *s:description* and *s:name*.[6]

There is a collection of properties, disjoint from types, in a multiple-parent generalization taxonomy with multiple roots. Each property is identified by a unique URL. The document (fragment) at that URL defines the property, providing:

1. types that its values belong to (its ranges), and
2. some properties that are more general than it (its parents).

Ranges and parents are the only information about a property obtainable from its defining document (fragment).

For each parent of the property for each range of the property the parent must have a range that is the same as or a generalization of the range. This condition on property ranges means that the validity of a property value can be checked by looking only at the range types of the property itself.

The properties *s:description* and *s:name* both have range *s:Text*.

Data values belong to one or more datatypes, and are disjoint from types and properties. Data values are written as a combination of a URL identifying a datatype and a

[5] Enumeration actually has a different supertype on *schema.org* but this account removes the unneeded supertype.

[6] There are several other properties for *s:Thing* on *schema.org*, but these do not play a role in this account and are ignored here.

text string. The mapping in the datatype turns the text string into a value of the datatype. Every data value belongs to *s:Datatype*. If a data value belongs to a datatype then it belongs to the parents of the datatype.

Items are things in the world, including information things, and are disjoint from types, properties, and data values. Items belong to (one or more) non-datatype types. Items have zero or more URLs identifying them. Items are associated with (other) items and data values via properties. Every item belongs to *s:Thing*. If an item belongs to a type then it belongs to the parents of the type.

If an item or data value is associated with an item via a property then the item or data value is also associated with the item via each parent of the property. For each item or data value associated with an item via a property,

1. one of the item's types has the property as one of its properties, and
2. the item or data value belongs to one of the ranges of the property.

The documents (document fragments) at the URLs identifying an item provide information about the item, including types for the item as well as items and data values associated with the item via properties.

Bare text can be used as if it was the value for any property. If the property does not have *s:Text* or *s:Datatype* as one of its ranges, but does have one or more datatypes as a range that have a data value that can be written as the bare text then the actual value for the property is one of these data values. If the property does not have *s:Text* or *s:Datatype* as one of its ranges, and does not have any suitable datatypes as a range, but does have one or more non-datatypes as a range, then the actual value for the property is some item that has a type that is one of these ranges and this item has the text as a value of its *s:description* property. (The property *s:description* is used instead of *s:name*, as the text might not truly be a name for the value.) Otherwise the actual value for the property is the bare text itself.

Any surface syntax must provide ways to write all possible data values (as long as they are not too big). Any surface syntax must have ways to provide items with any number of types, including none, and values for any property of any of the provided types or their generalizations or *s:Thing*, including allowing multiple values for a property. Any surface syntax must provide ways for writing items with no identifying URLs. Any surface syntax must specially process syntax that would otherwise produce values for *s:additionalType*, turning the values into types; and *s:url* and *s:sameAs*, turning the values into identifying URLs.

The following URLs are not used to identify types or properties: *s:Class*, *s:Property*, *s:domainIncludes*, *s:rangeIncludes*, *rdfs:subClassOf*, *rdfs:subPropertyOf*, *rdfs:domain*, *rdfs:range*, *rdfs:type*, *rdfs:Class*, *rdf:Property*, and *owl:Class*. If they are used in a surface syntax to provide information about an item they and their values must be ignored. The following URLs are not used to identify properties: *s:additionalType*, *s:url*, and *s:sameAs*.

Formal Definition for Schema.org

This definition for schema.org defines an abstract syntax for schema.org, abstracting away from the details of the various surface syntaxes, and a model-theoretic semantics, that provides a formal meaning for schema.org. It conforms to the pre-theoretic description above.

Abstract Syntax

Surface syntaxes for schema.org are transformed into an abstract syntax, in a process not fully described here. The abstract syntax plays a similar role as triples do for RDF [2], but is more complicated, as it makes distinctions between definitions and information about items. The abstract syntax removes artifacts of the surface syntaxes that make a formal account difficult, but transforming schema.org surface syntaxes into this abstract syntax is simple.

The gathering of information from Web documents is performed when building the abstract syntax. Constructs in the abstract syntax that start with a URL (or set of URLs) may be constructed from the document at the URL (or documents at one or more of the URLs), although they need not be.

Definition 1. *A* URL *in this document is a URL with optional fragment identifier, as defined in the W3C Working Draft on URLs [8]. A* text string *is a sequence of Unicode characters [9]. A* literal *is a pair consisting of a URL, the* datatype identifier *of the literal, and a text string.*

One part of schema.org information consists of definitions—of regular types, of enumerated types, of datatypes, and of properties.

Definition 2. *A* (regular) type definition *is a triple, $\langle U, S, P \rangle$, where U is a URL, the identifier of the type; S is a set of URLs, the supertypes of the type; and P is another set of URLs, the properties of the type.*

For example, the type definition for movies could be[7]

$$\begin{array}{l}\langle\ \textsf{\textit{s:Movie}}, \{\textsf{\textit{s:CreativeWork}}\}, \\ \quad \{\textsf{\textit{s:actor}}, \textsf{\textit{s:director}}, \textsf{\textit{s:producer}}, \textsf{\textit{s:duration}}, \textsf{\textit{s:musicBy}}, \\ \quad\ \ \textsf{\textit{s:productionCompany}}, \textsf{\textit{s:trailers:author}}, \textsf{\textit{s:copyrightYear}}\}\ \rangle\end{array}$$

indicating that movie is a subtype of creative work and has eight locally-specified properties.

The type of actions that update a collection would be

$$\langle \textsf{\textit{s:UpdateAction}}, \{\textsf{\textit{s:Action}}\}, \{\textsf{\textit{s:collection}}\}\rangle$$

indicating that update actions are actions and have only one locally-specified property.

[7] This definition of movies ignores the legacy properties for movies in *schema.org*, and adds in some properties from *s:CreativeWork* for illustrative purposes.

Definition 3. *An* enumerated type definition *is a quadruple, $\langle U, S, P, E\rangle$, where U, S, and P are as in a regular type definition, and E is yet another set of URLs, the direct instances of the enumerated type.*

For example, the enumerated type definition for book formats would be

$$\langle\ \textit{s:BookFormatType}, \{\textit{s:Enumeration}\}, \{\},\\ \{\textit{s:EBook}, \textit{s:Hardcover}, \textit{s:Paperback}\}\ \rangle$$

indicating that there are three different book formats.

Definition 4. *A* datatype definition *is a quadruple, $\langle U, S, W, I\rangle$, where U and S are as in a regular type definition, W is a set of values for the datatype, and I is a partial mapping from text strings into W.*

For example, the datatype for URLs would be

$$\left\langle \textit{s:URL}, \{\textit{s:Text}\}, U, id^U \right\rangle$$

where U is the set of Unicode strings that are valid URLs and id^U is the string identity function restricted to valid URLs, indicating that URL values are those text strings that are valid URLs.

It is a bit unusual to include formal datatype definitions in an abstract syntax. It would be more usual to pull these out into some sort of side definition. However, this way of defining datatypes puts all aspects of the definition in one place, as is done for other type and property definitions.

Definition 5. *A* property definition *is a triple, $\langle U, S, R\rangle$, where U is a URL, the identifier of the property; S is a set of URLs, the superproperties of the property; and R is a non-empty set of URLs, the ranges of the property.*

For example, the *s:collection* property (used in update actions), a subproperty of the *s:object* property with range *s:Thing*, would be written as the first of the following property definitions:

$$\langle \textit{s:collection}, \{\textit{s:object}\}, \{\textit{s:Thing}\}\rangle,\\ \langle \textit{s:actor}, \{\}, \{\textit{s:Person}\}\rangle,\\ \langle \textit{s:director}, \{\}, \{\textit{s:Person}\}\rangle,\\ \langle \textit{s:copyrightYear}, \{\}, \{\textit{s:Number}\}\rangle,\\ \langle \textit{s:author}, \{\}, \{\textit{s:Organization}, \textit{s:Person}\}\rangle.$$

Definition 6. *A* type definition *is either a regular type definition, an enumerated type definition, or a datatype definition. A* definition *is either a type definition or a property definition A definition is said to be a definition for its identifier.*

Type and property definitions define separate generalization partial orders on types and properties, building up from the supertypes and superproperties of type definitions and property definitions.

Definition 7. *A type (property) definition, D, is a* child *of another, D', if D' is one of the supertypes (superproperties) of D. A type (property) definition, D, is a* descendant *of another, D', in a set of definitions, if there is a chain of child relationships from D to D', in the set of definitions.*

Definition 8. *A URL, U, is a type (property)* descendant *of another, U', in a set of definitions, written $U < U'$, if U is the identifier of a type (property) definition that is a descendant of a type (property) definition that is identified by U'.*

The other part of schema.org information consists of information about items, providing types and property values for items. Note that this information is not called definitions, as there is no requirement that different items in the abstract syntax provide information about different resources.

Definition 9. *A* property value pair*, $\langle U, V \rangle$, consists of U, a URL identifying the property, and V, a text string or a literal or a URL or an item, indicating the value.*

Definition 10. *An* item *is a triple, $\langle N, T, PV \rangle$, where N is a (possibly empty) set of URLs, identifiers of the item; T is a (possibly empty) set of URLs, identifiers of types of the item; and PV is a (possibly empty) set of property value pairs.*

For example, an item for a particular movie could be

$\langle\, \{$*http://www.avatarmovie.com/index.html*, *w:Avatar_(2009_film)*$\}$,

$\{$*s:Movie*$\}$,

$\{\langle$*s:name*, *"Avatar"*$\rangle$,

$\langle$*s:director*, $\langle\, \{\}, \{$*s:Person*$\}, \{\langle$*s:name*, *"James Cameron"*$\rangle\}\ \rangle\ \rangle$,

$\langle$*s:actor*, *"Sam Worthington"*$\rangle$,

$\langle$*s:actor*, *w:Sigourney_Weaver*$\rangle\}$

$\langle$*s:year*, *"2009"*$\rangle$,

$\langle$*s:author*, *"James Cameron"*$\rangle\ \rangle$

This item has two identifiers and six property-value pairs, one providing a text value for a name of the movie, one providing an in-line item for a director of the movie, one providing a text value for an actor in the movie, one providing a URL identifying an actor in the movie, one providing a text value for a copyright year of the movie, and another providing a text value for an author of the movie.

A collection of definitions and items is a knowledge base, the overall way of collecting schema.org information together. There are many side conditions on schema.org knowledge bases to provide an overall structure of the generalization hierarchies for types and properties, to account for the built-in types and properties, and to ensure that literals are well behaved.

Definition 11. *A schema.org* knowledge base *is a triple, $\langle T, P, I \rangle$, where D is a set of type definitions, P is a set of property definitions, and I is a set of items that satisfies the following conditions:*

1. *Each URL is the identifier of at most one definition in T, and similarly for P. There is at most one definition for any URL in T and P.*
2. *The descendant relationship for types (properties) in T (P) is a strict partial order.*
3. *T contains the following regular type definitions:*
   $\langle \textsf{\textit{s:Thing}}, \{\}, \{\textsf{\textit{s:description}}, \textsf{\textit{s:name}}\}\rangle$,
   $\langle \textsf{\textit{s:Datatype}}, \{\}, \{\}\rangle$, *and*
   $\langle \textsf{\textit{s:Enumeration}}, \{\textsf{\textit{s:Thing}}\}, \{\}\rangle$.
4. *T contains the following datatype definition, where S is the set of text strings and id is the identity mapping on text strings:*
   $\langle \textsf{\textit{s:Text}}, \{\textsf{\textit{s:Datatype}}\}, S, id\rangle$
5. *P contains the following property definitions:*
   $\langle \textsf{\textit{s:description}}, \{\}, \{\textsf{\textit{s:Text}}\}\rangle$ *and* $\langle \textsf{\textit{s:name}}, \{\}, \{\textsf{\textit{s:Text}}\}\rangle$.
6. *For each literal $\langle U, V\rangle$ in the knowledge base there is a datatype definition $\langle U, S, W, I\rangle$ in T such that I is defined on V.*
7. *T has a datatype definition for U iff $U <$ s:Datatype in T.*
8. *T has an enumerated type definition for U iff $U <$ s:Enumeration in T.*
9. *T has a regular type definition for U iff U is s:Datatype or s:Thing or $U <$ s:Thing but not $U <$ s:Enumeration in T.*

There is nothing about web document retrieval in the abstract syntax (nor in the formal semantics immediately following). The *intent* should be clear, however—that type and property definitions come from the document (fragment) obtained by dereferencing the URL identifying the type or property and that item information often comes from the document (fragment) obtained by dereferencing a URL identifying the item.[8]

Building a schema.org knowledge base then generally starts with a collection of web documents that have schema.org markup about items. However, first the pages in *schema.org* that define types and properties are parsed to produce type and property definitions for the knowledge base. The document (fragments) accessible from URLs identifying items encountered during this parsing are added to the initial collection of web documents. Then this collection of documents is parsed to produce items for the knowledge base.

URLs for items that are encountered during the parsing may be used to direct that the web document (fragment) at that URL also be parsed to produce items for the knowledge base. Whether this “follow your nose” behavior is actually performed during the construction of a particular knowledge base depends on many factors that are outside the purview of this account.

Model-theoretic Semantics

The semantics for schema.org here is built up in the standard way from interpretations, which provide formal meanings for all URLs as identifying types or properties, and items and URLs as identifying resources. Items are mapped into sets of resources, not resources, as the resource corresponding to an item is indeterminate unless there is a URL that identifies the item. A single URL can independently identify a type, a property, and a resource, but these do not have any formal connection between them.

[8] Note that types and properties have a unique identifier whereas items may have multiple identifiers, or none.

Definition 12. *An* interpretation *is a sextuple, $\langle I_R, I_V, I_T, I_P, I_U, I_I\rangle$, where I_R is a set of resources; I_V is a set of data values, disjoint from I_R; and*

$$
\begin{aligned}
&I_T : U \to 2^{I_R} \cup 2^{I_V}\\
&I_P : U \to 2^{I_R \times (I_R \cup I_V)}\\
&I_U : U \to I_R\\
&I_I : I \to 2^{I_R}\\
&I_T(\textsf{\textit{s:Thing}}) = I_R\\
&I_T(\textsf{\textit{s:Datatype}}) = I_V
\end{aligned}
$$

where U is the set of URLs with optional fragments and I is the set of items.

I_T maps types into their extensions, a set of resources or a set of data values. I_P maps properties into their extensions, sets of pairs whose first element is a resource and whose second element is a resource or a data value. I_I maps items into their extensions, a set of resources. I_U maps URLs into their extensions as item identifiers, a resource;

Although the mappings above are infinite, in any knowledge base the only part of the mappings that are relevant are for the URLs and items that occur in the knowledge base (or query, in entailment and querying situations).

Definition 13. *An interpretation, $\langle I_R, I_V, I_T, I_P, I_U, I_I\rangle$, satisfies a knowledge base, $\langle K_T, K_P, K_I\rangle$ iff*

1. *for $\langle U, \{S_1, ..., S_n\}, P\rangle$ a regular type definition in K_T, $I_T(U) \subseteq I_T(S_i)$;*
2. *for $\langle U, \{S1, ..., Sn\}, P, \{E1, ...Em\}\rangle$ an enumerated type definition in K_T,*
   (a) $I_T(U) \subseteq I_T(S_i)$,
   (b) $I_T(U) = \{I_U(e) \mid e \in E\}$, and
   (c) $\forall e_i \neq e_j \in E \quad I_U(e_i) \neq I_U(e_j)$,
   where $E = \cup\{E' | \langle U', S', P', E'\rangle \in K_T \wedge (U' = U \vee U' < U)\}$;
3. *for $\langle U, \{S_1, ..., S_n\}, W, I\rangle$ a datatype definition in K_T,*
   (a) $I_T(U) \subseteq I_T(S_i)$ and
   (b) $I_T(U) = W$;
4. *for $\langle U, \{S1, ..., Sn\}, R\rangle$ a property definition in K_P, $\quad I_P(U) \subseteq I_P(S_i)$;*
5. *for $I = \langle\{U_1, ..., U_n\}, \{T_1, ..., T_m\}, \{\langle P_1, V_1\rangle, ..., \langle P_l, V_l\rangle\}\rangle$, an item in K_I,*
   (a) $I_I(I) = \{I_U(U_1)\}$, for $1 \leq i \leq n$,
   (b) $I_I(I) \subseteq I_T(T_i)$, $1 \leq i \leq m$,
   (c) for $x \in I_I(I)$, for $1 \leq i \leq l$, there exists $\langle P_i, S, R\}\rangle$ a property definition and there exists y such that $\langle x, y\rangle \in I_P(P_i)$ and
      - *if V_i is an item then $y \in I_I(V_i)$*
      - *if V_i is a URL then $y = I_U(V_i)$*
      - *if V_i is a literal $\langle D, T\rangle$ then there exists $\langle D, S, W, ID\rangle$ a datatype definition and $y = I_D(T)$*
      - *if V_i is a text string then*
        - *if* ***s:Text*** *$\in R$ or* ***s:Datatype*** *$\in R$ then $y = V_i$*
        - *otherwise if there exists $\langle D, S, W, I\rangle$ a datatype with D in R and V_i mapped by I then $y = I_T(V_i)$ for some one of these datatypes*
        - *otherwise if there exists $\langle T, S, P\rangle$ a type with T in R then $y \in I_T(T)$ for some one of these types, and $\langle y, V_i\rangle \in I_P(\textsf{\textit{s:description}})$*

- *otherwise* $y = V_i$;

6. *for each* U*, for each* $\langle x, y\rangle \in I_P(U)$*,*
   (a) *there exists* $\langle T, S, P\rangle$ *a type definition or* $\langle T, S, P, E\rangle$ *an enumerated type definition in* K_T *such that* $U \in P$ *and* $x \in I_T(U)$*, and*
   (b) *there exists* $\langle U, S, R\rangle$ *a property definition in* K_P *such that either* $R = \{\}$ *or there exists* $R' \in R$ *with* $y \in I_T(R')$.

From this basis the standard notions of entailment and inference and simple querying can be defined in the usual way.

The first clause in the satisfaction definition above provides the basis for the type generalization hierarchy, saying that the extension of a regular type is a subset of the extensions of each of its parent types. This is repeated for enumerated types and datatypes in the first part of the second and third clauses. Because regular and enumerated types are all descendants of *s:Thing*, their extensions are are subsets of the set of resources.

The second part of the second clause states that the extension of an enumeration is just the set of all the items that are stated to belong to it and its subtypes. The third part of the second clause states that all these items are different.

The third clause states that the extension of a datatype is its set of values. Because datatypes are all descendants of *s:Datatype*, their extensions must be subsets of the set of data values.

The fourth clause enforces the property generalization hierarchy.

The fifth clause handles all the parts of item syntax. The first part of this clause says that the extension of an item is the same as the extension of its identifiers, if any. Note that if there are no identifiers for an item, then the extension of the item need not be a singleton set. The second part says that the extension of an item is in the extension of the extensions of its types. The third part provides meaning for the property-value pairs in the item. There must be a property relationship from the item to a value that for item values is in the extension of the item, for URL values is the extension of the URL as an item identifier, and for literals is the correct data value. For values that are text there is a determination of what the most suitable ranges are with text datatypes the most suitable, other compatible datatypes next, and other types least suitable. Then one of these types is chosen and a data value or item is chosen to belong to this type.

So the movie item above would be a resource that is the same as the extensions of *http://www.avatarmovie.com/index.html* and *w:Avatar_(2009_film)* and is in the extension of *s:Movie*. This resource would be related to the string *"Avatar"* via (the property extension of) *s:name*, because *s:name* has *s:Text* as its sole range; to a resource in the extension of *s:Person* that has name *"James Cameron"* via *s:director*, to a resource in the extension of *s:Person* that has description *"Sam Worthington"* via *s:actor*, because *s:director* has *s:Person* as its sole range; to the extension of *w:Sigourney_Weaver* via *s:actor*; to the number 2009 via *s:copyrightYear*, because the supplied text is compatible with the *s:Number* datatype; and via *s:author* to a resource that is either in the extension of *s:Organization* or *s:Person* and that has description *"James Cameron"*, because *s:author* has ranges *s:Organization* and *s:Person*, and no datatype ranges.

The sixth clause enforces domain and range restrictions. The first part says that for each property, for each relationship between an item and a value in that property, there is a regular or enumerated type that has that property and contains the item. The second part similarly says that if the the definition of the property states ranges, then the value belongs to one of the ranges. Because parent properties have to have a range that is an ancestor of this range, this also satisfies the range restriction for each ancestor property.

Discussion

The above formal semantics is quite dense, particularly the definition of satisfaction. However, there is nothing particularly sophisticated going on, it is just that there are quite a few bits of schema.org markup to take into account.

The formal semantics is actually more standard than the formal semantics of RDF [10], as there are no resources for types and properties. It is easy to see that nothing about items can affect the relationships within and between types and properties (except that, as usual, inconsistencies in the information about items cause the semantics to collapse). If two URLs identify the same item, it is not necessarily the case that the two URLs define the same type or define the same property. This stands in stark contrast to RDF, but means that the only source of information for a type or property is its definition (which would come from the appropriate *schema.org* page). Thus consumers of schema.org information do not need to process any items to understand types and properties.

If two items share an identifying URL, then their extension is the same. If an item does not have a URL, but has a type that is an enumeration, then the item is one of finite, enumerated instances of the enumeration. This provides a weak form of disjunction for schema.org. The distinctness of the extensions of the URLs in an enumeration provides inequality for arbitrary resources in schema.org

As schema.org has weak disjunction and inequality for resources, its expressive power is considerably above that of RDFS, even though there is a translation provided from schema.org types and properties into RDFS.

Schema.org can, however, be translated into OWL, but the translation is not into a simple variant of OWL. For some parts of schema.org it is easy to see the translation. The special types *s:Thing* and *s:Datatype* are translated into *owl:Thing* and *owl:Literal*, respectively. All other non-datatype types become OWL classes. Supertypes where the parent type is a regular type translate into subtype axioms.

Property ranges for a property translate into a disjunctive property range axiom. However, a property can have both regular types and datatypes as ranges. A property with such a range cannot be categorized as either an object property or a data property, and so cannot be translated into OWL 2 DL. However, OWL 2 Full [11] permits uncategorized properties. Superproperties for properties then translate into subproperty axioms.

For some parts, the translation needs to take into account more than just one part of the knowledge base. Domains for a property are constructed by taking all the types that mention the property and producing a disjunctive property domain for the property from them. Enumerations are constructed by finding all the item URLs belonging to the enumeration type and its subtypes and constructing an axiom stating that the enumeration type is equivalent to the object one-of containing all these objects. This construction handles the supertype relationship where the supertype is an enumeration type. As well, a different individuals axiom is added stating that all the distinct URLs belonging to the enumeration type and its subtypes are different.

The translation for datatypes requires the construction of a datatype map that has the same effect as the datatype definitions. The datatypes in schema.org fit within what can be done for OWL datatypes, so this is possible. Supertypes for datatypes are either true, because the datatype's value spaces are in the correct relationship, or false. The first case can be ignored, as it has no effect. The second case can be translated into an inconsistent OWL assertion, as it produces an inconsistency.

The translation for items is a bit tricky, to allow for items that do not have any associated URLs. Anonymous individuals are employed in the translation to avoid the need for extra URLs, for each item there is generated a different anonymous individual.

For each identifier of the item, there is a same-as assertion between the anonymous individual and the identifier. For each type of the item, there is a class assertion stating that the anonymous individual belongs to the type.

Property value pairs are treated as follows.

1. If the value is a URL, then it identifies an item, and the URL is used directly as the value of a property assertion from the anonymous individual to the value via the property.
2. If the value is a literal, then the corresponding OWL literal is used instead in the property assertion.
3. If the value is an item, then the item is translated, and the anonymous individual for the item is used instead.
4. If the value is text, the situation is more complex. First then the ranges for the property are determined.
   (a) If *s:Text* or *s:Datatype* is one of the ranges then a string literal is constructed from the text and used as the value as above.
   (b) If there is a range that is a datatype with the text in the domain of its literal-to-value mapping, then those datatypes with the text in the domain of their literal-to-value mappings are used to construct one or more literals. The anonymous individual is then asserted to belong to a data some-value-from with the property to a data one-of constructed from these literals.
   (c) If there are no suitable datatype ranges then if there are any non-datatype ranges then the translation of the property-value pair is a property assertion from the anonymous individual via the property to a fresh anonymous individual. This fresh anonymous individual is asserted to belong to the disjunction of the non-datatype ranges and is also asserted to have the text as the value of the *s:description*.

The correctness of this mapping is not hard to verify, but a full proof of the correctness would be long and tedious and so is omitted here.

The translation into OWL does not determine how hard or easy reasoning is in schema.org because reasoning in OWL Full is undecidable. There are no inverse properties in schema.org, so not making the division between object and data properties does not appreciably affect reasoning. As this is the only part of the mapping that is not in OWL 2 DL, the mapping into OWL shows that the reasoning in schema.org is decidable.

It is easy to show that reasoning here is in PSpace, as reasoners need not introduce new types, properties, or items. Showing the precise complexity of reasoning is more difficult, as enumerations and the disjunctive nature of domain and range includes need to be addressed. For example, if this account of schema.org were modified to use the underlying semantics of RDF and lift the restrictions on the use of certain URLs the set cover problem could be encoded, introducing a new source of hardness to reasoning.

The intent of schema.org appears to be that all the types and properties defined in schema.org will remain in the *schema.org* namespace and thus under the control of the owners of *schema.org* and will change only infrequently. Other web pages will not be allowed to make changes or additions to these types and properties. This limits the effect of the non-monotonic nature of the disjunctive domains and ranges of schema.org.

The model-theoretic account here is a standard one, based on inference instead of constraints. If an item has a value for a property, then the item is inferred to belong to one of the domains of the property. The constraint reading [12] would instead require that the item be stated to belong to a domain for the property before a value could be provided. There are benefits to the constraint account, as it is closer to the database situation, but it is less flexible [13].

It is possible to get the effect of the constraint approach in a surface syntax. A surface syntax can have constructs that require that property-value pairs for an item be only for items that mention a type or subtype of one of the domains of the property. In this way most benefits of both approaches can be obtained.

Conclusion

This paper has provided an analysis of what schema.org should be, leading up to a complete formal treatment of schema.org including an abstract syntax and a model-theoretic semantics. It fills in voids in the publicly-available description of schema.org, including whether types and properties and items are disjoint, whether enumerations are distinct, whether properties can have generalizations, and how to handle text values. This may not exactly correspond to intent of the schema.org members, but it is consistent with the available information about schema.org, and uses only a reasonable set of additional assumptions.

This paper shows that even the unusual parts of schema.org can be translated into OWL. Although schema.org cannot be translated into OWL 2 DL, because schema.org properties cannot be categorized into object and data properties, the extensions are cosmetic, and schema.org reasoning is no harder than reasoning in OWL 2 DL. Determining just how hard schema.org reasoning is remains as further work.

Schema.org does not provide local ranges for properties, such as saying that the author of a movie is a person even though in general authors can be either people or organizations. This lack of expressive power limits what can be said about property values and is especially problematic as quite a few roles in *schema.org* could benefit from local ranges (e.g., a season of a TV series should be a TV season, but can only be a general season, as Radio Series also have seasons). Adding this feature to schema.org would usefully improve its expressive power.

The account of schema.org here should provide a starting point for further formal analysis of schema.org and a firm foundation for systems that consume schema.org information. Web pages that contain schema.org information can be checked against this account to provide a formal account of what the information conveys, thus reducing the possibility of mismatches between providers and consumers of schema.org information.

The obvious next step is to gather large amounts of schema.org information to see how schema.org is used in practice. This usage should then be analyzed to see how well the various aspects of schema.org are used and how the account here helps to better provide meaning for actual information that uses schema.org.

References

1. Schreiber, G., Raimond, Y.: RDF 1.1 primer. W3C Working Group Note, http://www.w3.org-/TR/rdf11-primer/ (25 February 2014)
2. Cyganiak, R., Wood, D., Lanthaler, M.: RDF 1.1 concepts and abstract syntax. W3C Recommendation, http://www.w3.org/TR/rdf-concepts/ (25 February 2014)
3. W3C OWL Working Group: OWL 2 Web Ontology Language: Document overview. W3C Recommendation, http://www.w3.org/TR/owl2-overview (11 December 2012)
4. Motik, B., Patel-Schneider, P.F., Parsia, B.: OWL 2 web ontology language: Structural specification and functional-style syntax. W3C Recommendation, http://www.w3.org/TR/owl2-syntax/ (11 December 2012)
5. Bollacker, K., Tufts, P., Pierce, T., Cook, R.: A platform for scalable, collaborative, structured information integration. In: Sixth International Workshop on Information Integration on the Web, Vancouver, British Columbia, AAAI Press (July 2007)
6. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F., eds.: The Description Logic Handbook: Theory, implementation, and applications. 2nd edn. Cambridge University Press (2007)
7. Birbeck, M., McCarron, S.: CURIE syntax 1.0: A syntax for expressing compact URIs. W3C Working Group Note, http://www.w3.org/TR/curie/ (2007)
8. Arvidsson, E., Smith, M.: URL. W3C Working Draft, http://www.w3.org/TR/URL/ (24 May 2012)
9. The Unicode Consortium: The unicode standard. http://www.unicode.org/versions/latest/ (2013)
10. Hayes, P., Patel-Schneider, P.F.: RDF 1.1 semantics. W3C Recommendation, http:-//www.w3.org/TR/rdf11-mt/ (25 February 2014)
11. Schneider, M.: OWL 2 web ontology language: RDF-based semantics (second edition). W3C Recommendation, http://www.w3.org/TR/owl-rdf-based-semantics/ (11 December 2012)
12. de Bruijn, J., Polleres, A., Lara, R., Fensel, D.: OWL DL vs. OWL Flight: Conceptual modeling and reasoning for the semantic web. In: Proceedings of the 14th World Wide Web Conference, Japan (May 2005)
13. Patel-Schneider, P.F., Horrocks, I.: Position paper: A comparison of two modelling paradigms in the semantic web. In: Proceedings of the 15th International Conference on the World Wide Web (WWW2006), New York, NY, ACM Press (May 2006)