A Test Suite for JSON Schema Containment*

Lyes Attouche[1], Mohamed-Amine Baazizi[2], Dario Colazzo[1], Yunchen Ding[3], Michael Fruth[3], Giorgio Ghelli[4], Carlo Sartiani[5], and Stefanie Scherzinger[3]

[1] Université Paris-Dauphine & Université PSL, CNRS, LAMSADE, Paris, France
{lyes.attouche, dario.colazzo}@dauphine.fr
[2] Sorbonne Université, LIP6 UMR 7606, France baazizi@ia.lip6.fr
[3] University of Passau, Passau, Germany
{michael.fruth, stefanie.scherzinger}@uni-passau.de
[4] Dipartimento di Informatica, Università di Pisa, Italy ghelli@di.unipi.it
[5] DIMIE, Università della Basilicata, Italy carlo.sartiani@unibas.it

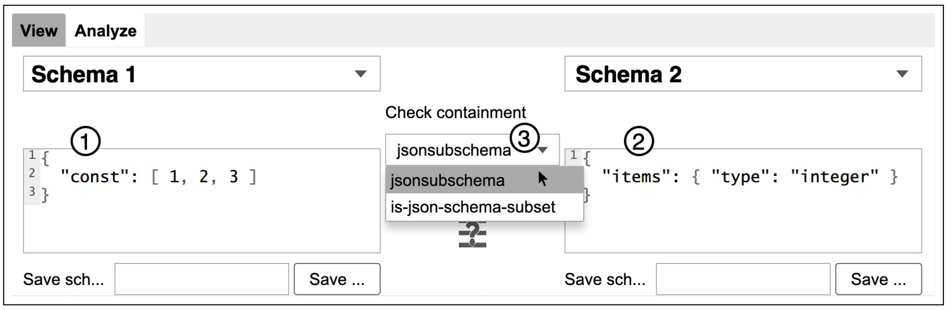
Abstract. JSON is a very popular data exchange format, and JSON Schema an increasingly popular schema language for JSON. Evidently, schemas play an important role in implementing conceptual models. For JSON Schema, there is a first generation of tools for checking whether one schema is contained in another. This is an important task when comparing schemas, and ultimately, the conceptual models that they capture. Testing whether such tool implementations are correct is difficult, since writing test cases requires a deep understanding of the JSON Schema language. In this demo, we present the first systematically generated test suite for JSON Schema containment checking. This test suite consists of pairs of schemas where the containment relationship is known *by construction*. Our test suite aims at covering all language features of JSON Schema. Applying existing containment checkers (including our own implementation) to our test suite, we discovered implementation bugs not known to us. We offer our test suite to the research community as well as to tool developers, hoping to contribute to the development of JSON Schema containment checkers.

Keywords: JSON Schema Containment Checking · Test Suite · Comparing Conceptual Models

1 Introduction

Nowadays, the most widely used data exchange format is JSON, thanks to its flexibility and its ability to represent both objects and sequences. JSON Schema is increasingly adopted for specifying and validating JSON instances. Software systems exposed as API rely on JSON Schema to express the shape of the

* This contribution was partly supported by the *Deutsche Forschungsgemeinschaft* (DFG, German Research Foundation) – 385808805.



```
{
  "const": [ 1, 2, 3 ]
}
```

```
{
  "items": { "type": "integer" }
}
```

Fig. 1. Screenshot showing test schemas side-by-side. We use the tool Josch [9] as an editor and for comparing the verdicts of different JSON Schema containment checkers.

expected request and the format of the returned data. Some machine learning libraries [4] also resort to JSON Schema to verify the input-output compatibility of the pipelines operators. Hence, it becomes paramount to develop tools for deciding whether the set of instances of one schema are included in the set of instances of another schema, that is, to check schema containment.

In the case of JSON, checking for schema containment can be particularly challenging, as JSON Schema [11], the *de-facto* standard schema language for JSON data [3], is extremely powerful, but also complex (especially when negation is involved [5]). As an example, consider the assertion `"items": {"type": "integer"}` in schema 2 from Figure 1 (②). This assertion states that the elements in an array can be integers only, but the assertion is also satisfied by any JSON value that is not an array.

When comparing two schemas, or two versions of some schema, a crucial question is how these schemas relate. For instance, in Figure 1, the left schema (here, an array of three integer constants) is contained in the right: Any instance valid w.r.t. the left schema is also valid w.r.t. the right schema, but not vice versa.

Syntax-driven approaches for checking schema containment cannot be easily adapted, nor tree-automata approaches, as used for XML types [14]. Indeed, JSON Schema is inherently more expressive than XML Schema [13], given the presence of the `uniqueItems` assertion; in a similar way, approaches based on stacks and regular expression inclusion algorithms cannot cope with the constraints of JSON Schema. More sophisticated approaches are required. First proposals have been made [2,10], yet given the lack of a trusted testbed, checking the correctness of such tools remains a challenge [8].

Our Contributions. We propose here a test suite for checking JSON Schema containment. Our test suite comprises pairs of schemas and the schema containment result. We made the deliberate decision to not hand-craft our test suite, or use real-world examples, as this would make it vulnerable to errors. Rather, we derive all tests from an existing ground truth, the JSON Schema Test Suite [12] (a collection of schema validation tests, which is a different and well-explored task). By design, our test suite covers all keywords in the JSON Schema language.

We have made our test suite available as open source. Our Zenodo archive at https://zenodo.org/record/5336931 also links to our GitHub repository.

2 Preliminaries

JSON data model. The grammar below captures the syntax of JSON values, which are basic values, objects, or arrays. Basic values B include the null value, booleans, numbers n, and strings s. Objects O represent sets of members, each member being a name-value pair, and arrays A represent sequences of values.

$J ::= B \mid O \mid A$		**JSON expressions**
$B ::= \text{null} \mid \text{true} \mid \text{false} \mid n \mid s$	$n \in \mathsf{Num}, s \in \mathsf{Str}$	**Basic values**
$O ::= \{l_1 : J_1, \ldots, l_n : J_n\}$	$n \geq 0, \quad i \neq j \Rightarrow l_i \neq l_j$	**Objects**
$A ::= [J_1, \ldots, J_n]$	$n \geq 0$	**Arrays**

JSON Schema. JSON Schema is a language for defining the structure of JSON documents. The syntax and semantics of JSON Schema have been formalized in [13] (following Draft-04), and we merely present the main keywords informally:

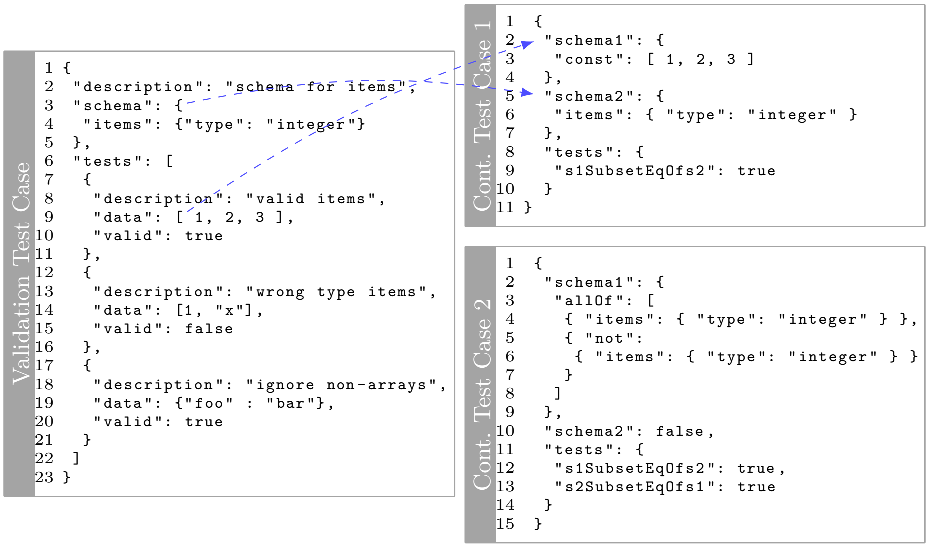
Assertions include `required`, `enum`, `const`, `pattern` and `type`, and indicate a test that is performed on the corresponding instance.

Applicators include the boolean operators `anyOf`, `allOf`, `oneOf`, `not`, the object operators `properties`, `patternProperties`, `additionalProperties`, the array operator `items`, and the reference operator `$ref`. They indicate a request to apply a different operator to the same instance or to a component of the current instance.

Annotations include `title`, `description`, and `$comment`, they do not affect validation but they indicate an annotation associated to the instance.

JSON Schema Validation. In validating a JSON instance against a schema, we check whether the instance is valid w.r.t. the schema. Implementations are available in most common programming languages [1]. The JSON Schema Test Suite [12] is a collection of tests for validators, covering the entire JSON Schema language. Figure 2 shows a sample test case on the left, which we discuss further below. Programmers write a script in their preferred programming language, to parse this JSON-encoded input, and to perform unit tests, e.g., as done in [7].

Example 1. We present the general idea behind the test case in Figure 2 (left). The schema to be tested starts in line 3. It declares that if the instance is an array, then all items must be integer-typed. The data instances to be tested are contained in lines 9, 14, and 19. Lines 10, 15, and 20 state whether the data is a valid instance w.r.t. the schema. Note that due to the conditional semantics of JSON Schema, the object in line 19 is indeed a valid instance, because the integer-type assertion only applies *if* the instance is an array.



```
1  {
2   "description": "schema for items",
3   "schema": {
4    "items": {"type": "integer"}
5   },
6   "tests": [
7    {
8     "description": "valid items",
9     "data": [ 1, 2, 3 ],
10    "valid": true
11   },
12   {
13    "description": "wrong type items",
14    "data": [1, "x"],
15    "valid": false
16   },
17   {
18    "description": "ignore non-arrays",
19    "data": {"foo" : "bar"},
20    "valid": true
21   }
22  ]
23 }
```

```
1  {
2   "schema1": {
3    "const": [ 1, 2, 3 ]
4   },
5   "schema2": {
6    "items": { "type": "integer" }
7   },
8   "tests": {
9    "s1SubsetEqOfs2": true
10  }
11 }
```

```
1  {
2   "schema1": {
3    "allOf": [
4     { "items": { "type": "integer" } },
5     { "not":
6      { "items": { "type": "integer" } }
7     }
8    ]
9   },
10  "schema2": false,
11  "tests": {
12   "s1SubsetEqOfs2": true,
13   "s2SubsetEqOfs1": true
14  }
15 }
```

Fig. 2. Left: Test case from the JSON Schema Test Suite [12] for testing JSON Schema validators (for Draft 6, adapted from file `items.json`, commit hash #fe94275). Right: Derived test cases for JSON Schema containment. Dashed arrows indicate data flow.

JSON Schema Containment. In the following, we write $S \subseteq S'$ to denote that schema S is contained in S', meaning that any JSON instance valid w.r.t. S is also valid w.r.t. S'. We write $S \equiv S'$ to denote schema equivalence, which can be checked by double inclusion. While JSON Schema validation can be checked in polynomial time, containment checking results to be EXPTIME-hard [6].

3 The JSON Schema Containment Test Suite

We next describe how we construct our test suite for checking JSON Schema containment, by programmatically deriving tests from the validation test suite. Our main approach is to use the Boolean operators `not` (for negation), `anyOf` (for disjunction), and `allOf` (for conjunction), to derive pairs of schemas such that we have a clear understanding of their containment relationship.

Example 2. Let us consider Figure 2. To the left, we see the test case of the JSON Schema Test Suite discussed earlier. To the right, we see two derived test cases: Containment Test Case 1 is an alternative encoding of the first validation test from the left: `schema1` declaring an array constant with three integer members (derived from line 9 on the left) is contained in `schema2` (derived from line 3 on the left). Line 9 states this relationship.

Test Case 2 is based on a different idea. Line 3 requires that two subschemas must be matched in a way that is unsatisfiable, the one in line 4 (taken from the validation test), and its negation. In line 10, we declare the unsatisfiable schema `false`. Here, `schema1` and `schema2` are equivalent (see lines 12 and 13). While this may seem obvious to human observers, it can be challenging for tools.

We next describe how we systematically derive our test cases.

Reflexivity. For each schema S, it holds that $S \equiv S$. While this might seem trivial, it is nevertheless a challenge for existing implementations [8].

Validation. For each valid instance v of a schema S, we encode the validation:

$$\texttt{\{ "const": } v \texttt{ \}} \quad \subseteq \quad S \tag{1}$$

$$\texttt{\{ "const": } v \texttt{ \}} \quad \not\subseteq \quad \texttt{\{ "not": } S \texttt{ \}} \tag{2}$$

Given all valid instances $v_1, \ldots, v_n$ from a validation test, we further derive

$$\texttt{\{ "anyOf": [\{ "const": } v_1 \texttt{ \}, ..., \{ "const": } v_n \texttt{ \}] \}} \quad \subseteq \quad S$$

Correspondingly, for each invalid instance i, we derive a test case

$$\texttt{\{ "const": } i \texttt{ \}} \quad \not\subseteq \quad S$$

Empty and universal language. Given a schema S, we derive an unsatisfiable schema, i.e., a schema equivalent to the schema `false`.

$$\texttt{\{ "allOf": [}S\texttt{, \{ "not": } S \texttt{ \}] \}} \quad \equiv \quad \texttt{false}$$

Similarly, we declare a schema that is universally satisfied (`true`), by replacing the conjunction `allOf` by the disjunction `anyOf`:

$$\texttt{\{ "anyOf": [}S\texttt{, \{ "not": } S \texttt{ \}] \}} \quad \equiv \quad \texttt{true}$$

Test case statistics. We provide test cases for all adopted drafts (six at the time of writing) of JSON Schema. Specifically, for Draft 6, we derive 2 120 schema pairs. In 60%, the first is a subset of the second. By not exclusively generating test cases where the first schema is contained in the second, containment checkers that always declare the first schema to be a subset of the second will fail in 40%.

Putting the Test Suite to the Test. We have used our test suite to assess the language coverage of our own containment checker [2], as well as of two preceding implementations [8]. For all tools, we found gaps in coverage. For our own tool [2], we identified a failure rate of 25%. Discussing the problematic schemas in detail is beyond the scope of this demo proposal. Overall, as the test suite was able to reveal actual problems, we find it vastly helpful for our purposes.

4 Demonstration Scenario

We next outline our planned demonstration scenario.

1. We introduce our audience to the JSON Schema language and its semantics. We illustrate how this language can capture complex conceptual models.
2. We review the JSON Schema Test suite, originally designed for validation experiments. We then show how we derive containment tests.
3. We engage our attendees in a mini game, where we employ our interactive schema management tool Josch [9][1]. We pre-load the test schemas in Josch, and show them side-by side, as seen in Figure 1 (① and ②).
   - We ask our attendees to vote (e.g., using an online polling tool) whether they regard one schema to be a subschema of the other.
   - We resort to existing tools for checking JSON Schema containment (③ in the screenshot), to compare the shown schemas. Our attendees will notice that there are tests where the tool implementations do not agree.
   - We reveal the answer to containment checking, based on our test suite.

   We plan our mini game as an entertaining and engaging way to quickly gain a certain level of understanding of the JSON Schema language, and to recognize the need for a well-principled schema containment test suite. We hope that by participating in our demo, the demo attendees will find inspiration for their own research in the conceptual modeling field.

Acknowledgments. We thank Luca Escher (University of Passau) for code quality assurance and Wolfgang Mauerer (Technical University of Applied Sciences Regensburg) for sharing the LaTeX template used to create Figure 2.

References

1. JSON Schema Validators. Available at: https://json-schema.org/implementations.html (2021)
2. Attouche, L., Baazizi, M.A., Colazzo, D., Falleni, F., Ghelli, G., Landi, C., Sartiani, C., Scherzinger, S.: A Tool for JSON Schema Witness Generation. In: Proc. EDBT. pp. 694–697 (2021)
3. Baazizi, M.A., Colazzo, D., Ghelli, G., Sartiani, C.: Schemas And Types For JSON Data. In: Proc. EDBT'19. pp. 437–439 (2019)
4. Baudart, G., Hirzel, M., Kate, K., Ram, P., Shinnar, A.: Lale: Consistent automated machine learning. In: KDD Workshop on Automation in Machine Learning (2020), https://arxiv.org/abs/2007.01977
5. Bazizi, M.A., Colazzo, D., Ghelli, G., Sartiani, C., Scherzinger, S.: An Empirical Study on the "Usage of Not" in Real-World JSON Schema Documents. In: Proc. ER (2021)
6. Bourhis, P., Reutter, J.L., Suárez, F., Vrgoc, D.: JSON: Data model, Query languages and Schema specification. In: Proc. PODS. pp. 123–135 (2017)
7. Ebdrup, A.: JSON Schema Benchmark. Available at: https://github.com/ebdrup/json-schema-benchmark, version of commit hash: #e9c884f. (2021)

[1] Note that our demo proposal does *not* feature Josch as its contribution, but our JSON Schema containment test suite. We merely use Josch as a convenient editor.

8. Fruth, M., Baazizi, M.A., Colazzo, D., Ghelli, G., Sartiani, C., Scherzinger, S.: Challenges in Checking JSON Schema Containment over Evolving Real-World Schemas. In: Proc. EmpER. pp. 220–230 (2020)
9. Fruth, M., Dauberschmidt, K., Scherzinger, S.: Josch: Managing Schemas for NoSQL Document Stores. In: Proc. ICDE. pp. 2693–2696 (2021)
10. Habib, A., Shinnar, A., Hirzel, M., Pradel, M.: Finding Data Compatibility Bugs with JSON Subschema Checking. In: ISSTA. p. 620–632 (2021)
11. json-schema-org: JSON Schema (2021), available at: https://json-schema.org
12. json-schema.org: JSON Schema Test Suite. Available online at https://github.com/json-schema-org/JSON-Schema-Test-Suite, commit hash: #09fd353. (2021)
13. Pezoa, F., Reutter, J.L., Suárez, F., Ugarte, M., Vrgoc, D.: Foundations of JSON Schema. In: Proc. WWW. pp. 263–273 (2016)
14. Tozawa, A., Hagiya, M.: XML schema containment checking based on semi-implicit techniques. In: CIAA. pp. 213–225 (2003)