ASON: A semantically complete superset of JSON

Version 0.3

Casey Dahlin

September 28, 2014

1 Introduction

JSON[1] has become an increasingly popular data interchange format for a wide variety of applications, from web services and RPC interfaces, to databases. Yet the JSON spec has never specified a semantic for JSON values.

Meanwhile, NoSQL databases, some of which use JSON for data presentation or more, are rising in popularity, yet there are no agreed upon semantics for querying such a database.

ASON is a superset of JSON syntax, as well as a semantic. The acronym stands for **A**lgebraic **S**erialized **O**bject **N**otation. It is designed to represent not only JSON values, but families of values. It includes a series of transforms which can allow a large, complex value or composition of values, such as might represent an entire table in a database, to be queried or made to conform to a schema.

It is hoped that ASON will serve not only these practical purposes, but also open the door to a formalization for the transformation of unstructured data, similar to what relational algebra has been for relational databases.

2 Additional Syntax

We will assume a general familiarity with JSON syntax already. To JSON's syntax, ASON adds three new invariant values, much like $true$, $false$, and $null$:

- The Universe value (U)
- The Wild value ($*$)
- The Empty value ($\emptyset$, also rendered as _)

These can appear anywhere that a JSON value could appear; as list items, values in an object, or by themselves.

[1] RFC 4627 - http://www.ietf.org/rfc/rfc4627.txt?number=4627

2.1 Operators

In addition to the new constants, ASON introduces four new operators; three binary infix operators and one unary prefix operator.

Operators and parenthesis behave as structural characters do in the JSON syntax[2].

The operators, in order of descending precedence, are:

- Complement (!) A prefix operator.
- Join (:)
- Intersect ($\cup$ or &)
- Union ($\cap$ or |)

2.2 Universal Object Syntax

ASON introduces a subtype of object called a "universal object." It is represented identically to a normal object, except that after the list of key-value pairs is one "," and one "$*$". An example:

$$\{\text{“}num\text{”}: \quad 6, \quad \text{“}string\text{”}: \quad \text{“}hello\text{”}, \quad *\} \tag{1}$$

Please note that we will typically use the term "object" to refer to either normal or universal objects.

3 ASON Intuitively

The purpose of this section is to give an intuitive grasp of ASON. The next section will endeavor to rigorously formalize it. For this reason, this section may gloss over or ignore several issues. Those implementing the specification should have a thorough understanding of the formal transformations in ASON.

ASON relies on a bijection between ASON values and sets. All ASON values which reflect typical singular JSON values map to sets of cardinality 1. We state that two ASON values are equal *if and only if* their mapped sets are equal.

3.1 Operations in ASON

We can think of ASON values as collections of JSON values, so the ASON value $\{\text{“}foo\text{”}: \quad 8\}$ is a set containing only the identical JSON value.

The purpose of the $\cup$ and $\cap$ operators is thus apparent: they combine and manipulate these singleton collections to form collections containing multiple values. Their specific function is upon the underlying sets, and analogous to the equivalent set theory operations: the union of two ASON values is an ASON value whose underlying set is the union of each of the values' underlying set. So the ASON value 6 is a collection containing 6, and the ASON value $6 \cup 7$ is a collection containing 6 and 7.

[2]RFC 4267, section 2

3.2 The Universe, Wild, and Empty Values

The additional values we have added over JSON can all be explained in these terms: $\emptyset$ is a value underlied by the empty set, and represents a collection containing nothing. U is the ASON universe, and is a collection of all possible ASON values. $*$ is similar to U, except it does not contain the value $null$.

3.3 Universal Objects

Universal objects are used to refer to collections of objects which all have certain keys in common. For example, the collection of all objects which have a key "foo" with value 6 would be:

$$\{\text{“}foo\text{”}: \quad 6, \quad *\} \tag{2}$$

3.4 Complementation

The ! operator also specifies a union of several values. Specifically it specifies a union of all values not equal to the value it is applied to.

It is intuitive to say the collection of all values which are not 6 is !6. It is possibly more subtle to point out $U = !\emptyset$, or $* = !null$, but these equalities should hold given our definition.

3.5 Join

Join operations can combine the keys of two objects, and also simulate the SQL join operation when applied to unions of objects.

4 Formalization

We will allow for numbers in JSON/ASON form to be transformed in ways that mathematics implies yield identical results, for example $6.0 = 6$ and $+3 = 3$. We will also assume that characters in strings may be replaced with their equivalent unicode escape sequences as specified by JSON, and vice versa. Given this and the transformations on other values we will outline below, we will define equality thus:

Rule 1. *Two ASON values are equal* if and only if *they can each be transformed into an identical representation.*

Reflexively, this specification shall describe a series of transformations on ASON values which do not alter their semantic value. We will also state that:

Rule 2. *If a value A can be transformed into a value B, then the value B may be transformed into the value A.*

And lastly:

Rule 3. *All operators are associative.*

4.1 Basic Transforms

These are intuitive rules understood by most existing JSON users and implementations. We will formalize them here.

Rule 4. *The order of two key-value pairs in an object may be exchanged.*

Rule 5. *Values within objects or lists may be transformed.*

4.2 Key Expansion

Rule 6.

$$\begin{aligned} & \{\text{“}foo\text{”}: \quad 6\} \\ = \quad & \{\text{“}foo\text{”}: \quad 6, \quad \text{“}bar\text{”}: \quad null\} \end{aligned} \tag{3}$$

A non-universal object may have a key-value pair with any key and value null inserted in to it.

Rule 7.

$$\begin{aligned} & \{\text{“}foo\text{”}: \quad 6, \quad *\} \\ = \quad & \{\text{“}foo\text{”}: \quad 6, \quad \text{“}bar\text{”}: \quad U, \quad *\} \end{aligned} \tag{4}$$

A universal object may have a key-value pair with any key and value U inserted in to it.

Key expansion dramatically simplifies the definition of other operations. Performing key expansion is often a crucial step in computing reduced ASON values. Do however note that key expansion implies that there is no difference between a key being set to *null* and a key not being present, at least within non-universal objects. This is a logical, but not necessarily expected semantic for JSON's *null* value.

Reversal of this transformation via rule 2 is typically referred to as *key contraction* .

4.3 Obliteration

Rule 8.

$$\begin{aligned} [7, \quad 8, \quad 9, \quad \emptyset, \quad 10] &= \emptyset \\ \{\text{“}foo\text{”}: \quad 6, \quad \text{“}bar\text{”}: \quad \emptyset\} &= \emptyset \end{aligned} \tag{5}$$

Any object or list containing a value which is equal to $\emptyset$ may be replaced with $\emptyset$

Obliteration is somewhat counterintuitive, but is required to simplify other operations. Without this transformation it might be provable that the underlying set of a non-universal object has a cardinality greater than 1. Consider our coming definition of intersection.

4.4 Intersection Reduction

Rule 9.

$$\begin{aligned} & \left(\{\text{“}foo\text{”}: \quad 6\} \cup \{\text{“}bar\text{”}: \quad 7\}\right) \quad \cap \quad \{\text{“}baz\text{”}: \quad 8\} \\ = \quad & \{\text{“}foo\text{”}: \quad 6\} \cap \{\text{“}baz\text{”}: \quad 8\} \quad \cup \quad \{\text{“}bar\text{”}: \quad 7\} \cap \{\text{“}baz\text{”}: \quad 8\} \end{aligned} \tag{6}$$

The intersect operator may be distributed over the union operator.

This isn't at all surprising, and if we've begun to intuit the analogy between ASON union and intersection and set theory's implementations of the same, it becomes obvious.

Rule 10.

$$\begin{array}{rcrcl} 6 & \cap & 7 & = & \emptyset \\ \{\text{“}foo\text{”}: \ \text{“}bar\text{”}\} & \cap & 7 & = & \emptyset \\ [4, \ 5, \ 6] & \cap & 7 & = & \emptyset \\ [4, \ 5, \ 6] & \cap & [7, \ 8] & = & \emptyset \\ [4, \ 5, \ 6] & \cap & \{\text{“}foo\text{”}: \ \text{“}bar\text{”}\} & = & \emptyset \end{array} \tag{7}$$

The intersection of any two values which are not equal, and of which neither can be expressed as a union of two or more non-equal, non-empty values, and where both values are not objects, or equally-lengthed lists, is $\emptyset$.

This is a necessarily complex way of specifying what is ultimately a simple concept: intersecting unequal values which are not collections of multiple values yields an empty value. It becomes muddled when we attempt to express it as purely transformation over syntax, as this specification aims to do.

Rule 11.

$$\begin{array}{rcrcl} 6 & \cap & 6 & = & 6 \\ \{\text{“}foo\text{”}: \ \text{“}bar\text{”}\} & \cap & \{\text{“}foo\text{”}: \ \text{“}bar\text{”}\} & = & \{\text{“}foo\text{”}: \ \text{“}bar\text{”}\} \\ [4, \ 5, \ 6] & \cap & [4, \ 5, \ 6] & = & [4, \ 5, \ 6] \end{array} \tag{8}$$

The intersection of any two values which are equal is equal to those values.

The counterpart to the previous rule is much more cleanly specified. A value intersected with itself is itself.

Rule 12.

$$[a, \ b, \ c] \ \cap \ [d, \ e, \ f] = [a \cap d, \ b \cap e, \ c \cap f] \tag{9}$$

The intersection of any two lists of the same length can be reduced to a list which is also that length and where each element is the intersection of the corresponding two elements in the operand lists.

This can almost be inferred from the previous rules and some of the properties of unions discussed later. In fact you can prove rules 11 and 10 for the specific case of lists of the same length with this rule. Nonetheless this rule does clarify the details of intersection.

Rule 13.

$$\begin{array}{rl} & \{\text{“}foo\text{”}: \ a, \ \text{“}bar\text{”}: \ b, \ \text{“}baz\text{”}: \ c \ \} \ \cap \\ & \{ \ \text{“}bar\text{”}: \ d, \ \text{“}baz\text{”}: \ null, \ \text{“}bam\text{”}: \ e \ \} \\ = & \{\text{“}foo\text{”}: \ a, \ \text{“}bar\text{”}: \ b, \ \text{“}baz\text{”}: \ c, \ \text{“}bam\text{”}: \ null \ \} \ \cap \\ & \{\text{“}foo\text{”}: \ null, \ \text{“}bar\text{”}: \ d, \ \text{“}baz\text{”}: \ null, \ \text{“}bam\text{”}: \ e \ \} \\ = & \left\{ \text{“}foo\text{”}: \ a \cap null, \ \text{“}bar\text{”}: \ b \cap d, \ \text{“}baz\text{”}: \ c \cap null, \ \text{“}bam\text{”}: \ e \cap null \right\} \end{array} \tag{10}$$

An intersection of two or more objects, universal or otherwise, can be replaced with a single object, where that object is universal if and only if both of the two operands were universal.

1. *Use key expansion so both original objects have the same set of keys. The result object will also have that set of keys*
2. *For each key, the result object will have the intersection of the two values for that key in the operands.*

This complements the previous rule, and rounds out our definition of intersection. Again, it nicely parallels rule 10, but does resolve real ambiguity in the system.

4.5 Join Reduction

Rule 14.

$$\begin{aligned} & \left(\{\text{“}foo\text{”}: \quad 6\} \cup \{\text{“}bar\text{”}: \quad 7\}\right) \quad : \quad \{\text{“}baz\text{”}: \quad 8\} \\ = \quad & \{\text{“}foo\text{”}: \quad 6\} : \{\text{“}baz\text{”}: \quad 8\} \quad \cup \quad \{\text{“}bar\text{”}: \quad 7\} : \{\text{“}baz\text{”}: \quad 8\} \end{aligned} \tag{11}$$

The join operator may be distributed over the union operator.

An issue of precedence, which should be well expected by now.

Rule 15.

$$\begin{array}{rcll} 6 & : & 7 & = \emptyset \\ \{\text{“}foo\text{”}: \quad \text{“}bar\text{”}\} & : & 7 & = \emptyset \\ [4, \quad 5, \quad 6] & : & 7 & = \emptyset \\ [4, \quad 5, \quad 6] & : & [7, \quad 8] & = \emptyset \\ [4, \quad 5, \quad 6] & : & \{\text{“}foo\text{”}: \quad \text{“}bar\text{”}\} & = \emptyset \\ & & & \\ 6 & : & null & = 6 \end{array} \tag{12}$$

The joining of any two values which are not equal, and of which neither can be expressed as a union of two or more non-equal, non-empty values, is $\emptyset$, unless one of those values is null, in which case the join may be expressed as the other value singly.

We borrow this from the intersect operator, which join mirrors, and add an exception, which wholly encapsulates how join differs from intersect.

Rule 16.

$$\begin{array}{rcll} 6 & : & 6 & = 6 \\ \{\text{“}foo\text{”}: \quad \text{“}bar\text{”}\} & : & \{\text{“}foo\text{”}: \quad \text{“}bar\text{”}\} & = \{\text{“}foo\text{”}: \quad \text{“}bar\text{”}\} \\ [4, \quad 5, \quad 6] & : & [4, \quad 5, \quad 6] & = [4, \quad 5, \quad 6] \end{array} \tag{13}$$

The join of any two values which are equal is equal to those values.

Here intersect behaves identically.

Rule 17.

$$[a, \quad b, \quad c] \quad : \quad [d, \quad e, \quad f] = [a : d, \quad b : e, \quad c : f] \tag{14}$$

The join of any two lists of the same length can be reduced to a list which is also that length and where each element is the join of the corresponding two elements in the operand lists.

Again, similar to intersection.

Rule 18.

$$\begin{array}{rlllllllll} & \{\text{“}foo\text{”}: & a, & \text{“}bar\text{”}: & b, & \text{“}baz\text{”}: & c & & & \} \\ & \{ & & \text{“}bar\text{”}: & d, & \text{“}baz\text{”}: & null, & \text{“}bam\text{”}: & e & \} \\ = & \{\text{“}foo\text{”}: & a, & \text{“}bar\text{”}: & b, & \text{“}baz\text{”}: & c, & \text{“}bam\text{”}: & null & \} \\ & \{\text{“}foo\text{”}: & null, & \text{“}bar\text{”}: & d, & \text{“}baz\text{”}: & null, & \text{“}bam\text{”}: & e & \} \\ = & \left\{\text{“}foo\text{”}: \right. & a : null, & \text{“}bar\text{”}: & b : d, & \text{“}baz\text{”}: & c : null, & \text{“}bam\text{”}: & e : null & \left.\right\} \end{array} \quad (15)$$

A join of two or more objects, universal or otherwise, can be replaced with a single object, where that object is universal if and only if both of the two operands were universal.

1. *Use key expansion so both original objects have the same set of keys. The result object will also have that set of keys*
2. *For each key, the result object will have the join of the two values for that key in the operands.*

We finish here the same way we finish in intersection.

4.6 Union Reduction

Rule 19.

$$a \cup \emptyset = a \quad (16)$$

Any value unioned with $\emptyset$ can be reduced to that value singly.

This more or less defines $\emptyset$. Combined with rule 8, it means that $\emptyset$ should almost always be able to be removed from an ASON value by transformation.

Rule 20.

$$a \cup a = a \quad (17)$$

Any value unioned with itself can be reduced to that value singly.

Interestingly, this parallels intersection.

Rule 21.

$$\begin{array}{rcl} [a, \quad b, \quad c] \cup [a, \quad b, \quad d] & = & [a, \quad b, \quad c \cup d] \\ \{\text{“}foo\text{”}: \quad a, \quad \text{“}bar\text{”}: \quad b\} \cup \{\text{“}foo\text{”}: \quad a, \quad \text{“}bar\text{”}: \quad c\} & = & \{\text{“}foo\text{”}: \quad a, \quad \text{“}bar\text{”}: \quad b \cup c\} \end{array} \quad (18)$$

Any union of lists or objects where the operands differ in only one value may be reduced to a version of that list or object where the differing value is a union of the values within the two operands.

This rule is key, but it is more often used in reverse. Unions can be propagated upward such that there is only one top-level sequence of unions by repeatedly exchanging values containing unions for unions of values.

4.7 Complementation

Rule 22.

$$!!a \quad = \quad a \tag{19}$$

Complementation of a complemented value is that value.

Rule 23.

$$!a \cap a \quad = \quad \emptyset \tag{20}$$

Intersecting a value with its complement yields $\emptyset$.

Rule 24.

$$!(a \cup b) \quad = \quad !a \cap !b \tag{21}$$

A complement of a union of two values is equal to the intersection of the complement of those two values separately.

Complementation is defined by a series of simple rules: it is its own inverse, complemented values don't intersect with the original values, and complementing a union is equivalent to intersecting the complements of its operands (an analogue to DeMorgan's law).

4.8 Constants

$$\begin{aligned} U &= !\emptyset \\ * &= !null \end{aligned} \tag{22}$$

With complementation defined, the two constant values are easily expressed. In fact they are revealed as syntactic sugar.

5 The Representation Relationship

Representation is intuitively simple. It is a relationship between ASON values, rendered as $\subseteq$ or the keyword *in*, which states that all values represented by the left operand are represented by the right (but not necessarily the other way around).

Formally we say that:

$$a \subseteq b \tag{23}$$

When and only when:

$$a \cap b = a \tag{24}$$

It is worth noting that $\emptyset$ Is represented in all valid ASON values by this logic.

6 Orders

One of the most important reasons to transform an ASON value is to place it in the simplest possible form. We divide ASON values into a series of numbered *orders*, which roughly identify how complex the simplest form of the value is.

Each order is a superset of all the orders below it, so an order 0 value is also an order 1 value. However, when we refer to "the order of value B" we typically refer to the minimally-applicable order. For the purposes of this section, we will denote variables with their implied order as a subscript, so variables referring to order 0 values will appear as a_0 and soforth.

6.1 Closure

An important property of each order is *closure*. An order is *closed* under a given operator if applying that operator to two values of that order yields a value also of that order. For example, if $a_n \cap b_n$ is always a value of order n then order n is closed under intersection.

In addition to the four ASON operators, we will also discuss whether values are closed under *containment*, that is, does using a value of a given order as the only element of a list or object also yield a value of that order. If so, that order is closed under containment.

6.1.1 Closure and Join

Because the join operator is equivalent to the intersect operator in most cases, closure under intersection should imply closure under join in all such cases. The excepted case is joining with the *none* value, which will always be closed, because $a_n : none = a_n$, which is order n. Therefore we can ignore closure under join and simply discuss closure under intersection.

6.2 Order 0

Order 0 is the simplest order. We define that for any values a and b such that:

$$a \subseteq b_0 \tag{25}$$

The value b_0 is order 0 if and only if one of the following is true:

$$\begin{aligned} a &= b_0 \\ a &= \emptyset \end{aligned} \tag{26}$$

It should be noted that $\emptyset$ trivially satisfies both requirements and is thus order 0.

6.2.1 Closure under containment

Rule 8 proves that containment of $\emptyset$ must result in $\emptyset$, so containing $\emptyset$ is closed.

Containment for other order 0 values is fairly easy to see. We will prove it here only for lists, though a similar proof works for objects as well.

Consider a_0 to be an order 0 object. Let's assume that $[a_0]$ is not an order 0 object. That means there must be some value c that satisfies all three of the following:

$$\begin{aligned} &c \subseteq [a_0] \\ &c \neq [a_0] \\ &c \neq \emptyset \end{aligned} \tag{27}$$

We can substitute our definition of representation from section 5 to get:

$$\begin{aligned} &c = c \cap [a_0] \\ &c \neq [a_0] \\ &c \neq \emptyset \end{aligned} \tag{28}$$

Rules 10 and 12 imply that anything we intersect with a list of length 1 must also be a list of length 1, or the result will be $\emptyset$, so we can rewrite c as follows:

$$\begin{aligned} &[d] = [d] \cap [a_0] \\ &[d] \neq [a_0] \\ &[d] \neq \emptyset \end{aligned} \tag{29}$$

Rule 12 also lets us rewrite the first premise:

$$\begin{aligned} &[d] = [d \cap a_0] \\ &[d] \neq [a_0] \\ &[d] \neq \emptyset \end{aligned} \tag{30}$$

We know from the definition of the order that $[d \cap a_0]$ must be either $[\emptyset]$ or $[a_0]$. In the first case, rule 8 would make the whole value equal to $\emptyset$, violating the third premise. In the second case we violate the second premise plainly.

6.2.2 Closure under intersection

The proof of closure under intersection for order 0 very nearly restates the definition of the order. The value of $a_0 \cap b_0$ must be represented in both a_0 and b_0, which limits it to being equal to them both, or $\emptyset$.

6.3 Order 1

Order 1 is any value expressable in the form:

$$a_0 \cup b_0 \tag{31}$$

Or the form:

$$a_1 \cup b_1 \tag{32}$$

6.3.1 Closure under union

Closure under union is implied in the definition of the order.

6.3.2 Closure under containment

Suppose one has an object containing a value that is order 1, which is to say it is a union of multiple order 0 values. One can split that object by rule 21 into multiple objects each of which has an order 0 value in that position, and which are joined externally by a union. If the objects have no other order 1 values then they are themselves order 0 and the union of them is order 1. If they have other order 1 values we can apply the process recursively until they no longer do and we are left with a chained union of several order 0 values.

6.3.3 Closure under intersection

Rule 9 states we can distribute an intersection across unions. This means we can continue to distribute intersections until we have a union of intersections of order 0 values. Order 0 is closed under intersection so the result is simply a union of order 0 values, which is order 1.

6.4 Order 2

Order 2 contains any value expressable as:

$$!a_1 \tag{33}$$

6.4.1 Closure under union

Closure under union is implied in the definition of the order.

6.4.2 Closure under complement

Given an order 1 value a_1, $!a_1$ must be order 2, but $!!a_1$ is order 1, because the complementation can be eliminated given rule 22.

If an order 2 value is a union of other order 2 values, then by associativity we can eventually express it as unions of a series of order 1 values, and a series of complemented order 1 values. Example:

$$a_1 \cup b_1 \cup c_1 \cup !d_1 \cup !e_1 \cup !f_1 \tag{34}$$

We can use rule 24 and rule 3 to isolate the inverted items and convert them to an intersection:

$$a_1 \cup b_1 \cup c_1 \cup !(d_1 \cap e_1 \cap f_1) \tag{35}$$

Order 1 is closed under union and intersection, so we can substitute:

$$g_1 \cup !h_1 \tag{36}$$

Now let's complement this form:

$$!(g_1 \cup !h_1) \tag{37}$$

We can apply rule 24 again to get:

$$!g_1 \cap h_1 \tag{38}$$

h_1 is either an order 0 value or a union of one or more such values by definition:

$$!g_1 \cap (i_0 \cup j_0 \cup k_0) \tag{39}$$

We can apply rule 9 to distribute:

$$!g_1 \cap i_0 \cup !g_1 \cap j_0 \cup !g_1 \cap k_0 \tag{40}$$

Order 0 values can't be reflected by a union of one or more non-empty, non-equal values, so the result of the above intersections must be just the order 0 values by rule 11 or $\emptyset$ by 10. The result is a union of order 0 values, or an order 1 value.

6.4.3 Closure under intersection

Suppose the form:

$$a_2 \cap b_2 \tag{41}$$

We can rewrite this in the form:

$$!(!a_2 \cup !b_2) \tag{42}$$

Order 2 is closed under complement and union, so the two variables are rewritable as uncomplemented order 2 values:

$$!(c_2 \cup d_2) \tag{43}$$

Order 2 is also closed under union:

$$!e_2 \tag{44}$$

And since, again, we are closed under complement, this is an order 2 value.

6.4.4 Order of U and $*$

Since *null* and $\emptyset$ are order 0, and complements of order 0 values are order 2, $!null$ and $!\emptyset$, also representable as U and $*$, are order 2.

6.5 Order 3

Order 3 contains all remaining values. This includes the universal object $*$, and values containing order 2 values.

6.6 Order and "Cardinality"

One way to look at orders is to look at them in terms of "cardinality." An order 0 value has a cardinality of 1, which is to say it represents exactly one JSON value. Order 1 values can represent finite collections of JSON values, Order 2 values can represent infinite collections of values, but their complements always represent finite collections of values. Order 3 contains values which represent infinite sets of values, but which may be complemented to represent finite sets of values.