1 Operations, Primitives and Algorithms

The following sections introduce, define and explain Operations, Primitives and Algorithms generally using the Terminology presented below. Operations are the building blocks of Primitives whereas Primitives are the building blocks of Algorithms. The definitions which follow are flexible enough to support implementation across programing languages but have been inspired by the core concepts found within Lisp. The focus of these sections is to define the properties of and interactions between Operations, Primitives and Algorithms in a general way which doesn't place unnecessary bounds on their range of possible functionality with respect to processing xAPI data.

1.1 Terminology

In the subsections and sections which follow, (s) indicates one or more

1.1.1 Scalar

Singular value x of a fundamental JSON type as described by JSON Schema

1.1.2 Collection

a n-tuple of items x such that

$$X = < x_i..x_n..x_j >$$

where

$$i \leq n \leq j \Rightarrow i \prec n \prec j \iff i \neq n \neq j$$

1.1.3 Key

A lookup k for a v within a kv where $k = x \vee X$

1.1.4 Value

a piece of data v where $v = x \vee X$

1.1.5 Key Value pair

Association between a k and v where

$$k \mapsto v$$

such that

$$kv = k \mapsto v$$

and a collection of Key Value pair(s) is defined as

$$KV = < kv_i..kv_n..kv_j >$$

such that

$$k_n \mapsto v_n$$

and all k within KV are unique

$$i_k \neq n_k \neq j_k$$

but the same is not true for all v within KV

$$i_v = n_v \vee i_v \neq n_v \;\; i_v = j_v \vee i_v \neq j_v \;\; j_v = n_v \vee j_v \neq n_v$$

1.1.6 Statement

Immutable collection of Key Value Pair(s) conforming to the xAPI Specification as described in the previous section

1.1.7 Algorithm State

Mutable collection of Key Value Pair(s)

1.1.8 Option

Collection of Key Value Pair(s) which alter the result of an Algorithm

2 Operation

Given an input X, an Operation produces output Y

If X changes to X' then the Operation results in Y' instead of Y

2.1 Domain

Any of the following

- Key(s)
- Value(s)
- Key Value pair(s)
- Statement(s)
- Algorithm State

2.2 Range

Any of the following dependent upon the Domain and Functionality of the Operation

- Key(s)
- Value(s)
- Key Value pair(s)
- Statement(s)
- Algorithm State

2.3 Formal Definition

A relationship between input and output data which will result in the same Y given the same X

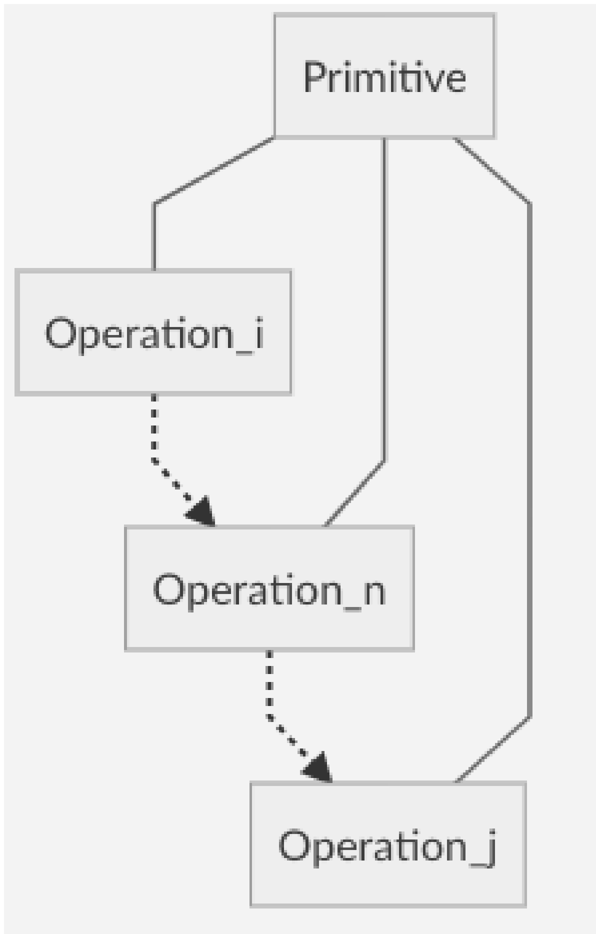
$$Operation(X) = Y \wedge \ Operation(X') = Y'$$

$$\Rightarrow$$

$$Y = Y' \iff X = X' \wedge Y \neq Y' \iff X \neq X'$$

3 Primitive

A collection of Operations where the output of an Operation is passed as the argument to the next Operation



Primitives break the processing of xAPI data down into discrete units that can be composed to create new analytical functions. Primitives allow users to address the methodology of answering research questions as a sequence of generic algorithmic steps which establish the necessary data transformations, aggregations and calculations required to reach the solution in an implementation agnostic way.

3.1 Domain

Any of the following

- Key(s)
- Value(s)
- Key Value pair(s)
- Statement(s)
- Algorithm State

3.2 Range

Any of the following dependent upon the Domain and Functionality of the Primitive

- Key(s)
- Value(s)
- Key Value pair(s)
- Statement(s)
- Algorithm State

3.3 Formal Definition

A collection of Operation(s) O_n labeled p and defined as

$$p =< O_i..O_n..O_j >$$

where

$$i \leq n \leq j \Rightarrow i \prec n \prec j \iff i \neq n \neq j$$

such that the output Z is defined as the sequential composition of operation(s) O_n given arg(s) $Args$ provided to p

$$Z = p(Args) = O_j(O_n(O_i(Args)))$$

4 Algorithm

Given a collection of statement(s) $S_{<a..b..c>}$ and potentially option(s) *opt* and potentially an existing Algorithm State *state* an Algorithm A executes as follows

1. call *init*
2. for each $stmt \in S_{<a..b..c>}$
   (a) *relevant*?
   (b) *accept*?
   (c) *step*
3. return *result*

with each process within A is enumerated as

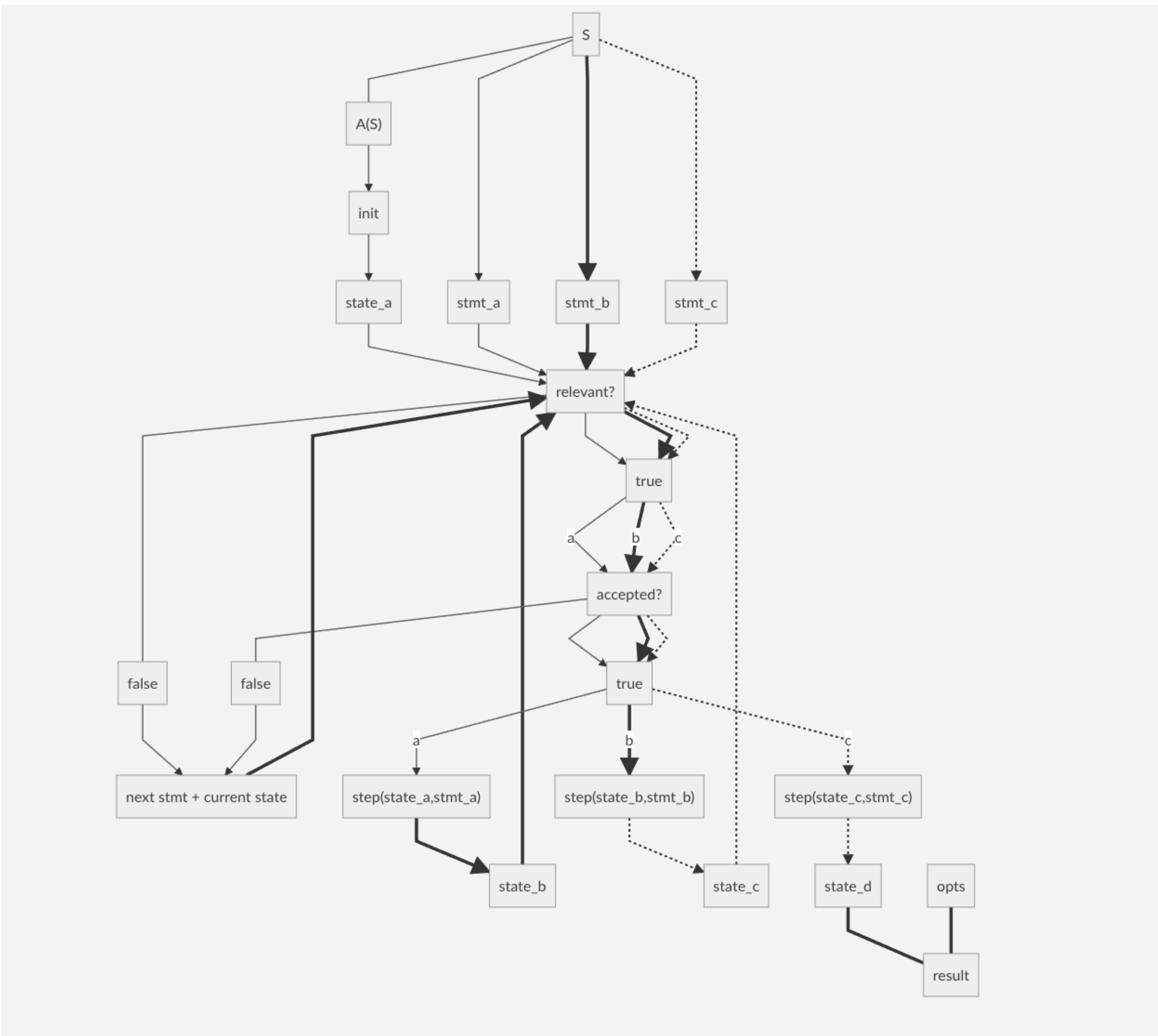
```
(init [state] body)
 - init state

(relevant? [state statement] body)
 - is the statement valid for use in algorithm?

(accept? [state statement] body)
 - can the algorithm consider the current statement?

(step [state statement] body)
 - processing per statement
 - can result in a modified state

(result [state] body)
 - return without option(s) provided
 - possibly sets default option(s)

(result [state opts] body)
 - return with consideration to option(s)
```

where

- *body* is a collection of Primitive(s) P which establishes the processing of inputs $\rightarrow$ outputs
- *state* is a mutable collection of key value pair(s) KV and synonymous with Algorithm State
- *statement* is a single statement s within the collection of statements S passed as input data to the Algorithm A

- *opts* are additional arguments passed to the algorithm A which impact the return value of the algorithm

Such that the execution of A can be described visually but not exhaustively as



4.1 Domain

Any of the following

- Statement(s)
- Algorithm State
- Option(s)

4.2 Range

- Algorithm State

4.3 Initialization

First process to run within an Algorithm which returns the starting Algorithm State $state_0$

$$init() = init(state) \vee init() \neq init(state)$$

where $state_0$ does not need to be related to its arguments

$$init() \rightarrow state_0$$

but $state_0$ can be derived from some other *state* passed as an argument to *init*

$$init(state) \rightarrow state_0'$$

such that

$$state_0 \neq state_0'$$

but this functionality is dependent upon the *body* of an Algorithms' *init*

4.3.1 Domain

- Algorithm State

4.3.2 Range

- Algorithm State

4.4 Relevant?

First process that each *stmt* passes through such that

$$relevant? \prec accept? \prec step$$

resulting in an indication of whether the *stmt* is valid for use within algorithm A

$$relevant?(state, stmt) = true \ \vee false$$

The criteria which determines validity of *stmt* within A is defined by the *body* of $relevant?$

4.4.1 Domain

- Statement
- Algorithm State

4.4.2 Range

- Scalar

4.5 Accept?

Second process that each *stmt* passes through such that

$$relevant? \prec accept? \prec step$$

resulting in an indication of whether the *stmt* can be sent to *step* given the current *state*

$$accept?(state, stmt) = true \ \vee false$$

The criteria which determines usability of *stmt* given *state*‘ is defined by the *body* of $accept?$

4.5.1 Domain

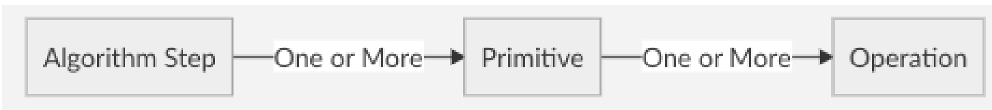
- Statement
- Algorithm State

4.5.2 Range

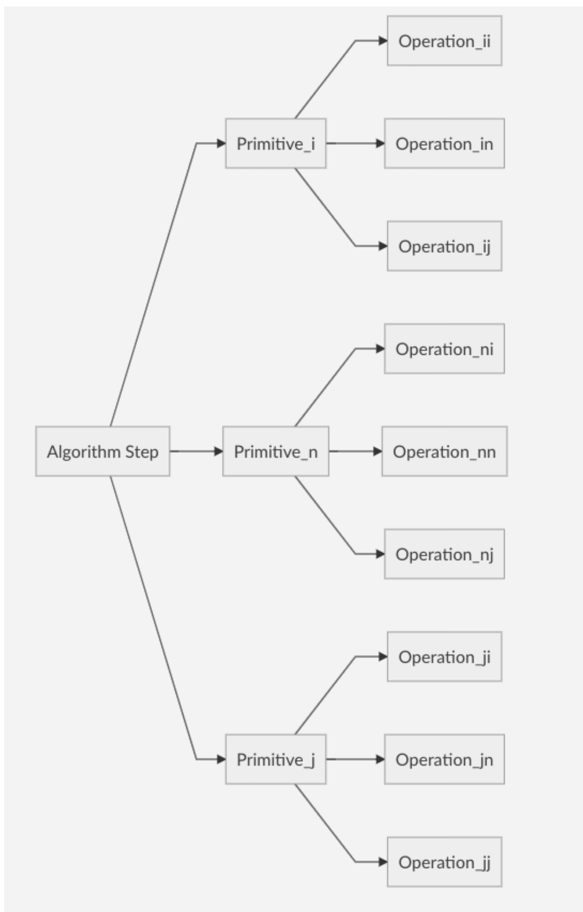
- Scalar

4.6 Step

An Algorithm Step consists of a collection of Primitive(s) and therefore collection(s) of Operation(s)



which expands to



$$i \leq n \leq j \Rightarrow i \prec n \prec j$$

$$i_i \leq i_n \leq i_j \leq n_i \leq n_n \leq n_j \leq j_i \leq j_n \leq j_j \Rightarrow i_{<i..n..j>} \prec n_{<i..n..j>} \prec j_{<i..n..j>}$$

where the output of a Primitive is passed as the argument to the next Primitive

The selection and ordering of Operation(s) and Primitive(s) into an Algorithmic Step determines how the Algorithm State changes during iteration through Statement(s) passed as input to the Algorithm.

4.6.1 Domain

- Statement
- Algorithm State

4.6.2 Range

- Algorithm State

4.6.3 Formal Definition

A collection of Primitive(s)

$$P =< p_i..p_n..p_j >$$

where

$$i \leq n \leq j \Rightarrow i \prec n \prec j \iff i \neq n \neq j$$

and

$$Z_i = p_i(Args) \Rightarrow O_{ij}(O_{in}(O_{ii}(Args)))$$

where

$$ii \leq in \leq ij \Rightarrow ii \prec in \prec ij \iff ii \neq in \neq ij$$

such that for each $stmt_b$ within a collection of Statement(s) S defined as

$$S =< stmt_a..stmt_b..stmt_c >$$

where

$$a \leq b \leq c \Rightarrow a \prec b \prec c \iff a \neq b \neq c$$

and

$$a \not\mapsto i \;\wedge b \not\mapsto n \;\wedge c \not\mapsto j$$

The output of *step* given a $stmt_b$ and $state_b$ is defined as

$$step(state_b, stmt_b) = p_j(p_n(Z_{ib}))$$

where

$$Z_{ib} = p_i(Args) \Rightarrow p_i(state_b, stmt_b) \Rightarrow O_{ij}(O_{in}(O_{ii}(state_b, stmt_b)))$$

and subsequently

$$Z_{nb} = p_n(Z_{ib})$$

which establishes that

$$Z_{jb} = p_j(Z_{nb}) \Rightarrow p_j(p_n(p_i(state_b, stmt_b)))$$

such that for a given $stmt_b$, $P_{<i..n..j>}$ will always result in a Z_{jb} but

$$Z_{ib} = state_b \vee state'_{ib} \iff state_b \neq state'_{ib}$$

which means

$$Z_{nb} = p_n(state_b, stmt_b) \vee p_n(state'_{ib}, stmt_b)$$
$$\Rightarrow$$
$$Z_{nb} = state_b \;\vee state'_{ib} \;\vee state'_{nb}$$
$$\Rightarrow$$
$$Z_{nb} = Z_{ib} \vee state'_{nb} \iff state_b \neq state'_{ib} \neq state'_{nb}$$

and concludes with

$$Z_{jb} = p_j(state_b, stmt_b) \vee p_j(state'_{ib}, stmt_b) \vee p_j(state'_{nb}, stmt_b)$$
$$\Rightarrow$$
$$Z_{jb} = state_b \vee state'_{ib} \vee state'_{nb} \vee state'_{jb}$$
$$\Rightarrow$$
$$Z_{jb} = Z_{nb} \vee state'_{jb} \iff state_b \neq state'_{ib} \neq state'_{nb} \neq state'_{jb}$$

such that

$$Z_{jb} \equiv state'_b$$

$$\Rightarrow$$

$$state'_b = state_b \vee state'_{ib} \vee state'_{nb} \vee state'_{jb} \iff state_b \neq state'_{ib} \neq state'_{nb} \neq state'_{jb}$$

the impact being that iteration through all $stmt \in S < a..b..c >$ results in a return of Z_{jc} such that

$$Z_{ja} = step(state_a, stmt_a) \Rightarrow state'_a \equiv Z_{ja} = state_a \vee state'_{ia} \vee state'_{na} \vee state'_{ja}$$

and

$$Z_{jb} = step(Z_{ja}, stmt_b) \Rightarrow state'_b \equiv Z_{jb} = Z_{ja} \vee state'_{ib} \vee state'_{nb} \vee state'_{jb}$$

meaning

$$Z_{jc} = step(Z_{jb}, stmt_c) \Rightarrow state'_c \equiv Z_{jc} = Z_{jb} \vee state'_{ic} \vee state'_{nc} \vee state'_{jc}$$

such that each $stmt \in S < a..b..c >$ may not result in a mutation of $state$ from $state \rightarrow state'$

$$state'_c = Z_{jc}$$

$$\Rightarrow$$

$$state'_c = state_a \vee state_{ia} \vee state'_{na} \vee state'_{ja} \vee state'_{ib} \vee state'_{nb} \vee state'_{jb} \vee state'_{ic} \vee state'_{nc} \vee state'_{jc}$$

$$\Rightarrow$$

$$state'c = state_a \vee state'c \neq statea$$

The no-op scenario described above is only a possibility of $step(state_a, stmt \in S_{<a..b..c>})$ but can be predicted to occur given

- The definition of individual Operations O which constitute a Primitive p

$$Operation(X) = Y \wedge Operation(X') = Y' \Rightarrow Y = Y' \iff X = X'$$

- The ordering of $O_{<i..n..j>}$ within p

$$i \prec n \prec j$$

- The Primitive(s) p chosen for inclusion within $P_{<i..n..j>}$

$$Z_i = p_i(Args) \Rightarrow O_{ij}(O_{in}(O_{ii}(Args)))$$

$$Z_j = p_j(Args) \Rightarrow O_{jj}(O_{jn}(O_{ji}(Args)))$$

$$\forall Args \; \exists \; Z_i = Z_j \iff O_{ij}(O_{in}(O_{ii}(Args))) \equiv O_{jj}(O_{jn}(O_{ji}(Args)))$$

$$< p_i, p_j > \equiv < p_j, p_i > \iff Z_i = Z_j$$

- The ordering of $p \in P_{i..n..j}$ which implies the ordering of $O \in p_{<i..n..j>} \in P_{<ii..ij..ni..nj..ji..jj>}$

$$i \prec n \prec j \Rightarrow ii \prec in \prec ij \Rightarrow ii \prec ij \prec ni \prec nj \prec ji \prec jj$$

$$P_{i..n..j} = P_{x..y..z} \Rightarrow < p_i, p_n, p_j > \equiv < p_x, p_y, p_z > \iff p_i \equiv p_x \land p_n \equiv p_y \land p_j \equiv p_z$$

$$\Rightarrow$$

$$P_{i..n..j} = P_{x..y..z} \iff i \mapsto x \land n \mapsto y \land j \mapsto z \land Z_i = Z_x \land Z_n = Z_y \land Z_j = Z_z$$

- The Key Value pair(s) $kv \in stmt \in S_{<a..b..c>}$
- The ordering of Statement(s) $stmt \in S_{<a..b..c>}$ such that $a \prec b \prec c$

4.7 Result

Last process to run within an Algorithm which returns the Algorithm State *state* without preventing subsequent call of A

$$relevant? \prec accept? \prec step \prec result \prec relevant? \iff S \neq \emptyset$$

$$\Rightarrow$$

$$relevant? \prec accept? \prec step \prec result \iff S = \emptyset$$

such that if $S(t_n) = \emptyset$ and at some future point j within the timeline $i..n..j$ this is no longer true $S(t_j) \neq \emptyset$ then

$$A(state_{n-1},\ S(t_{n-1})) = state_n = A(init(),\ S(t_{n-i})) \iff A(state_n,\ S(t_n)) = state_n$$

such that the statement(s) added to S between t_i and t_n is

$$S(t_{n-i})$$

and the statement(s) added to S between t_n and t_j be

$$S(t_{j-n})$$

such that

$$S(t_{n-i}) \ \cup \ S(t_{j-n}) = S(t_{j-i})$$

which means

$$A(init(),\ S(t_{j-i})) = state_j$$

and establishes that A can pick up from a previous $state_n$ without losing track of its own history.

$$A(result(state_n),\ S(t_{j-n})) = A(init(),\ S(t_{j-i})) = state_j$$

$$\Longleftrightarrow$$

$$result(state_n) = A(init(),\ S(t_{n-i})) = state_n$$

Which makes A capable of taking in some $S_{<i..n..j..\infty>}$ as not all $s \in S_{<i..\infty>}$ have to be considered at once. In other words, the input data does not need to persist across the history of A, only the effect of s on $state$ must be persisted.

Additionally, the effect of $opts$ is determined by the $body$ within $result$ such that

$$A(result(state_n),\ S(t_{j-n}),\ opts)$$

$$\equiv$$

$$A(init(),\ S(t_{j-i}))$$

$$\equiv$$

$$A(init(),\ S(t_{j-i}),\ opts)$$

$$\equiv$$

$$A(result(state_n),\ S(t_{j-n}))$$

Which implies that $opts$ may have an effect on $state$ but not in a way which prevents backwards compatibility of $state$

4.7.1 Domain

- Algorithm State
- Option(s)

4.7.2 Range

- Algorithm State