There will be many Primitives used within Algorithm definitions in DAVE but navigation into a nested *Collection* or *KV* is most likely to be used across nearly all Algorithm definitions. Because of this, the first common Primitive to be introduced is *walk*. In order to define *walk* using the Operation *recur*, the following helper Operations are introduced. These two helper Operations will be used to describe the navigation into and then back out of a nested Value based on the provided Collection of identifiers.

$$
\begin{array}{|l}
\hline
GetNext[V, Collection] \\
in?, next! : V \\
id? : Collection \\
getNext\_ : V \times Collection \nrightarrow V \\
\hline
next! = getNext(in?, id?) \bullet \\
\quad = (atIndex(in?, head(id?)) \iff (array?(in?) = true) \land (head(id?) \in \mathbb{N})) \lor \\
\quad\quad (atKey(in?, head(id?)) \iff (array?(in?) = false) \land (map?(in?) = true)) \\
\hline
\end{array}
$$

- Navigation down into either a *Collection* or *KV* based on the type of *in*?

$$
\begin{array}{|l}
\hline
Merge[(V, V), Collection] \\
parent?, child?, parent! : V \\
at? : Collection \\
merge\_ : (V \times V) \times Collection \rightarrowtail V \\
\hline
parent! = merge((parent?, child?), at?) \bullet \\
\quad = (associate(parent?, head(at?), child?) \\
\quad\quad \iff map?(parent?) = true) \lor \\
\quad\quad (update(parent?, child?, head(at?)) \\
\quad\quad\quad \iff (array?(parent?) = true) \land (head(at?) \in \mathbb{N})) \\
\hline
\end{array}
$$

- Updating of *parent*? to include *child*? at location indicated by *head*(*at*?)

The helper Operations defined above are necessary for describing the traversal of a heterogeneous nested Value. Collection and KV have different Fundamental Operations for navigation and update. Their usage in *walk* is touched on within the following summary and expanded further within the formal definition.

1. navigate down into the provided value *in*? up until the second to last value $in?_{path?_{j-1}}$ as described by the provided *path*?

$$
\begin{array}{l}
in?_{path?_{j-1}} : V \\
\vdash \\
path?_{j-1} \Rightarrow path? \lhd j \Rightarrow path? \lhd (\operatorname{dom} path? \setminus \{j\})
\end{array}
$$

2. extract any existing data mapped to $atIndex(path?, j)$ from the result of step 1

$$
\begin{array}{l}
in?_{path?} : V \\
\vdash \\
path? \Rightarrow path?_{j-1} \cup (j, atIndex(path?, j))
\end{array}
$$

3. create the mapping $(atIndex(path?\,, j), in?_{path?})$ labeled here as $args?$

$$\begin{array}{l} args? = (atIndex(path?\,, j), in?_{path?}) \\ \vdash \\ args? \in in?_{path?_{j-1}} \\ first(args?) = atIndex(path?\,, j) \end{array}$$

4. pass $args?$ to the provided function $fn?$ to produce some output $fn!$

$$fn! = fn?\,(args?) = fn?\,(atIndex(path?\,, j), in?_{path?})$$

5. replace the previous mapping $args?$ within $in?_{path?_{j-1}}$ with $fn!$ at $atIndex(path?\,, j)$

$$\begin{array}{l} child_j = first(args?) \mapsto fn! \\ in!\,?_{path?_{j-1}} = merge((in?_{path_{j-1}}, fn!), first(args?)) \\ \vdash \\ child_j \in in!\,?_{path?_{j-1}} \\ child_j \notin in?_{path?_{j-1}} \iff child_j \neq args? \\ args? \in in?_{path?_{j-1}} \\ args? \notin in!\,?_{path?_{j-1}} \iff args? \neq child_j \end{array}$$

6. retrace navigation back up from $in!\,?_{path?_{j-1}}$, updating the mapping at each $path?_n \in path?$ without touching any other mappings.

$$\begin{array}{l} in!\,?_{path?_{j-1}} \lhd first(args?) = in?_{path?_{j-1}} \lhd first(args?) \iff args? \neq child_j \\ \vdash \\ args? \neq child_j \Rightarrow second(args?) \neq second(child_j) \\ in!\,?_{path?_{j-1}} \lhd first(args?) \Rightarrow in!\,?_{path?_{j-1}} \lhd (\,\mathrm{dom}\ in!\,?_{path?_{j-1}} \setminus first(args?)) \end{array}$$

7. return $out!$ after the final update is made to $in?$.

$$\begin{array}{l} child_i = atIndex(path?\,, i) \mapsto in!\,?_{path?_i} \\ in!\,?_{path?_i} = merge((in?_{path?_i}, in!\,?_{path?_{i+1}}), atIndex(path?\,, i+1)) \\ \vdash \\ out! = merge((in?\,, second(child_i)), first(child_i)) \bullet \\ \qquad in? \lhd head(path?) = out! \lhd head(path?) \Rightarrow \\ \qquad\qquad \forall (a, b) \in path? \bullet b = atIndex(path?\,, a) \mid \exists\, a \bullet in?_a = out!_a \iff a \neq head(path?) \end{array}$$

The summary of *walk* given above is formalized within the schema $Walk$ bellow where $Walk$ dives deeper into the properties/constraints provided for each step. The variables names used in the summary are NOT used in all cases within $Walk$.

$$
\begin{array}{l}
\textit{Walk}[V, \textit{Collection}, (\_ \nrightarrow \_)] \\
\hline
\textit{GetNext}, \textit{Merge}, \textit{Recur} \\
\textit{in?}\,, \textit{out!}\,, \textit{fn!} : V \\
\textit{path?} : \textit{Collection} \\
\textit{fn?} : (\_ \nrightarrow \_) \\
\textit{walk}\_ : V \times \textit{Collection} \times (\_ \nrightarrow \_) \rightarrowtail V \\
\hline
\textit{walk} = \langle \langle \textit{getNext}\_, \textit{recur}\_ \rangle^{\#\,\textit{path?}-1}, (\_ \nrightarrow \_), \langle \textit{merge}\_, \textit{recur}\_ \rangle^{\#\,\textit{path?}-1} \rangle \\
\textit{out!} = \textit{walk}(\textit{in?}\,, \textit{path?}\,, \textit{fn?}\,) \bullet \\
\quad \forall n : i \,..\, j-1 \bullet j = \textit{first}(\textit{last}(\textit{path?}\,)) \Rightarrow \\
\qquad\qquad\qquad\qquad \textit{first}(j, \textit{path?}_j\,) \mid \exists\, \textit{down}_n \bullet \\
\qquad \textit{let}\ \ \textit{path?}_n == \textit{tail}(\textit{path?}\,)^{n-i} \\
\qquad\quad \textit{down}_i == \textit{getNext}(\textit{in?}\,, \textit{path?}_n\,) \Rightarrow \\
\qquad\qquad\qquad\quad \textit{atIndex}(\textit{in?}\,, \textit{head}(\textit{path?}\,)) \lor \textit{atKey}(\textit{in?}\,, \textit{head}(\textit{path?}\,)) \iff n = i \\
\qquad\quad \textit{down}_n == \textit{recur}(\textit{down}_i, \textit{path?}_n\,, \textit{getNext}\_)^{j-1} \\
\qquad\quad \textit{down}_{j-1} == \textit{getNext}(\textit{down}_n, \textit{path?}_n\,) \iff n = j-2 \\
\qquad \textit{down}_j = \textit{fn!} = \textit{fn?}\,(\textit{down}_{j-1}, \textit{path?}_n\,) \iff n = j-1 \Rightarrow \textit{path?}_n = \textit{path?}\,\restriction j \\
\\
\quad \forall z : p \,..\, q \bullet ((p = j-1) \land (q = i+1)) \Rightarrow \\
\qquad\qquad\qquad\qquad ((z = p \iff n = j-1) \land (z = q \iff n = i+1)) \mid \exists\, \textit{up}_n \bullet \\
\qquad \textit{let}\ \ \textit{path?}_{rev} == \textit{rev}(\textit{path?}\,) \\
\qquad\quad \textit{path?}_z == \textit{tail}(\textit{path?}_{rev}\,)^{p-z+1} \\
\qquad\quad \textit{up}_p == \textit{merge}((\textit{down}_{j-1}, \textit{down}_j), \textit{path?}_z\,) \Rightarrow \\
\qquad\qquad\qquad (\textit{path?}_z \equiv \textit{tail}(\textit{path?}_{rev}\,)) \land \\
\qquad\qquad\qquad\quad (\textit{associate}(\textit{down}_{j-1}, \textit{head}(\textit{path?}_z\,), \textit{down}_j) \lor \\
\qquad\qquad\qquad\quad \textit{update}(\textit{down}_{j-1}, \textit{down}_j, \textit{head}(\textit{path?}_z\,))) \iff z = p \\
\qquad\quad \textit{up}_z == \textit{recur}((\textit{down}_n, \textit{up}_p), \textit{path?}_z\,, \textit{merge}\_)^p \iff p = n+1 \land z = n \\
\qquad\quad \textit{up}_q == \textit{merge}((\textit{down}_{i+1}, \textit{up}_z), \textit{path?}_z\,) \iff z = q+1 \Rightarrow z = i+2 \\
\qquad\quad \textit{up}_i == \textit{merge}((\textit{down}_i, \textit{up}_q), \textit{path?}_z\,) \iff z = q \Rightarrow z = i+1 \Rightarrow \textit{up}_i = \textit{up}_{q-1} \\
\textit{out!} = \textit{merge}((\textit{in?}\,, \textit{up}_i), \textit{path?}_n\,) \iff (n = i = q-1) \Rightarrow \\
\qquad = \textit{merge}((\textit{in?}\,, \textit{merge}((\textit{down}_i, \textit{up}_q), \textit{tail}(\textit{path?}\,))), \textit{path?}\,) \\
\hline
\end{array}
$$

The following examples demonstrate the functionality of the Primitive *walk*

$$
\begin{array}{l}
X = \langle x_0, x_1, x_2 \rangle \ \land\ \textit{fn!} = \textit{fn}(\textit{val?}\,, \textit{idx?}\,) = \textit{ZZZ} \\
\qquad x_0 = \textit{true} \\
\qquad x_1 = \langle a, b, c \rangle \\
\qquad x_2 = \langle\!\langle \textit{foo} \mapsto \langle\!\langle \textit{bar} \mapsto \textit{buz}, x \mapsto y \rangle\!\rangle \rangle\!\rangle \\
\textit{walk}(X, \langle 0 \rangle, \textit{array?}\ \_) = \langle \textit{false}, x_1, x_2 \rangle \qquad\qquad [\textit{true} \neg \textit{Collection}] \\
\textit{walk}(X, \langle 2, \textit{foo}, z \rangle, \textit{fn}\_) = \langle x_0, x_1, \langle\!\langle \textit{foo} \mapsto \langle\!\langle \textit{bar} \mapsto \textit{buz}, x \mapsto y, z \mapsto \textit{ZZZ} \rangle\!\rangle \rangle\!\rangle \rangle \\
\textit{walk}(X, \langle 2, \textit{foo}, x \rangle, \textit{fn}\_) = \langle x_0, x_1, \langle\!\langle \textit{foo} \mapsto \langle\!\langle \textit{bar} \mapsto \textit{buz}, x \mapsto \textit{ZZZ} \rangle\!\rangle \rangle\!\rangle \rangle \\
\textit{walk}(X, \langle 2, \textit{qux} \rangle, \textit{fn}\_) = \langle x_0, x_1, (x_2 \cup \textit{qux} \mapsto \textit{ZZZ}) \rangle \\
\textit{walk}(X, \langle 1 \rangle, \textit{map}(\textit{succ}\_, x_1, 1)) = \langle x_0, \langle b, c, d \rangle, x_2 \rangle \\
\textit{walk}(X, \langle 1, 0 \rangle, \textit{succ}\_) = \langle x_0, \langle b, b, c \rangle, x_2 \rangle
\end{array}
$$