GRAFS: Graph Analytics Fusion and Synthesis Appendix

ANONYMOUS AUTHOR(S)

CONTENTS

1 Use-case Specifications

$$
\begin{aligned}
\mathrm{SSSP}(s)(v) &= \min_{p\in\mathsf{Paths}(s,v)} \mathsf{weight}(p) && \text{Shortest Path}\\
\mathrm{NP}(s)(v) &= |\,\mathsf{Paths}(s,v)\,| && \text{Number of Paths}\\
\mathrm{LP}(s)(v) &= \max_{p\in\mathsf{Paths}(s,v)} \mathsf{weight}(p) && \text{Longest Path}\\
\mathrm{SL}(s)(v) &= \min_{p\in\mathsf{Paths}(s,v)} \mathsf{length}(p) && \text{Shortest Length}\\
\mathrm{LL}(s)(v) &= \max_{p\in\mathsf{Paths}(s,v)} \mathsf{length}(p) && \text{Longest Length}\\
\mathrm{WP}(s)(v) &= \max_{p\in\mathsf{Paths}(s,v)} \mathsf{capacity}(p) && \text{Widest Path}\\
\mathrm{NP}(s)(v) &= \min_{p\in\mathsf{Paths}(s,v)} \mathsf{capacity}(p) && \text{Narrowest Path}\\
\mathrm{FR}(s)(v) &= \bigvee_{p\in\mathsf{Paths}(s,v)} \mathsf{True} && \text{Forward Reachability}\\
\mathrm{CC}(v) &= \min_{p\in\mathsf{Paths}(v)} \mathsf{head}(p) && \text{Connected Components}\\
\mathrm{CCS}(v) &= \bigcup_{p\in\mathsf{Paths}(v)} \{\,\mathsf{head}(p)\,\} && \text{Connected Component Set}\\
\mathrm{BR}(s)(v) &= \bigvee_{p\in\mathsf{Paths}(v,s)} \mathsf{True} && \text{Backward Reachability}\\
\mathrm{BFS}(s)(v) &= \mathsf{penultimate}(\operatorname*{arg\,min}_{p\in\mathsf{Paths}(s,v)} \mathsf{length}(p)) && \text{Breadth-First Search}
\end{aligned}
$$

Fig. 1. Use-cases for $\underset{p\in P}{\mathcal{R}} f(p)$ and $f(\arg \underset{p\in P}{\mathcal{R}}\, f(p))$

$$\mathrm{WSP}(s)(v) = \begin{aligned}[t] &\text{let } P := \operatorname*{args\,min}_{p \in \mathsf{Paths}(s,v)} \mathsf{length}(p) \text{ in} \\ &\max_{p \in P} \mathsf{capacity}(p) \end{aligned}$$

Widest Shortest Paths

$$\mathrm{NSP}(s)(v) = \left| \operatorname*{args\,min}_{p \in \mathsf{Paths}(s,v)} \mathsf{weight}(p) \right|$$

Number of Shortest Paths

$$\mathrm{HLP}(v) = \mathsf{head}\Big(\operatorname*{arg\,max}_{p \in \mathsf{Paths}(v)} \mathsf{weight}(p)\Big)$$

Head of Longest Path

$$\mathrm{HLL}(v) = \mathsf{head}\Big(\operatorname*{arg\,max}_{p \in \mathsf{Paths}(v)} \mathsf{length}(p)\Big)$$

Head of Longest Length

$$\mathrm{HNP}(v) = \mathsf{head}\Big(\operatorname*{arg\,min}_{p \in \mathsf{Paths}(v)} \mathsf{capacity}(p)\Big)$$

Head of Narrowest Path

$$\mathrm{SWSL}(s)(v) = \begin{aligned}[t] &\text{let } P := \operatorname*{args\,min}_{p \in \mathsf{Paths}(s,v)} \mathsf{length}(p) \text{ in} \\ &\min_{p \in P} \mathsf{weight}(p) \end{aligned}$$

Shortest Weight in
Shortest Length Paths

$$\mathrm{WSLSW}(s)(v) = \begin{aligned}[t] &\text{let } P := \operatorname*{args\,min}_{p \in \mathsf{Paths}(s,v)} \mathsf{weight}(p) \text{ in} \\ &\text{let } P' := \operatorname*{args\,min}_{p \in P} \mathsf{length}(p) \text{ in} \\ &\max_{p \in P'} \mathsf{capacity}(p) \end{aligned}$$

Widest in
Shortest Length in
Shortest Weight Paths

$$\mathrm{LNP}(s)(v) = \begin{aligned}[t] &\text{let } P := \operatorname*{args\,min}_{p \in \mathsf{Paths}(s,v)} \mathsf{capacity}(p) \text{ in} \\ &\max_{p \in P} \mathsf{length}(p) \end{aligned}$$

Longest Narrowest Path

$$\mathrm{HNP}(v) = \begin{aligned}[t] &P := \operatorname*{args\,min}_{p \in \mathsf{Paths}(v)} \mathsf{capacity}(p) \text{ in} \\ &\bigcup_{p \in P} \{\, \mathsf{head}(p) \,\} \end{aligned}$$

Heads of Narrowest Paths

$$\mathrm{CCSS}(v) = \left| \bigcup_{p \in \mathsf{Paths}(v)} \{\, \mathsf{head}(p) \,\} \right|$$

Connected Component Set Size

Fig. 2. Use-cases for nested $\underset{p \in P}{\mathcal{R}} f(p)$, part 1

$$\begin{aligned}
\mathrm{NWSP}(s)(v) \quad &= \qquad \qquad \text{Number of Widest Shortest Paths}\\
&\quad \text{let } P := \operatorname*{args\,min}_{p \in \mathsf{Paths}(s,v)} \mathsf{weight}(p) \text{ in}\\
&\quad \text{let } P' := \operatorname*{args\,max}_{p \in P} \mathsf{capacity}(p)\\
&\quad |\,P'\,|
\end{aligned}$$

$$\begin{aligned}
\mathrm{NSWSL}(s)(v) \quad &= \qquad \qquad \begin{array}{r}\text{Number of}\\ \text{Shortest Weight in}\\ \text{Shortest Length Paths}\end{array}\\
&\quad \text{let } P := \operatorname*{args\,min}_{p \in \mathsf{Paths}(s,v)} \mathsf{length}(p) \text{ in}\\
&\quad \text{let } P' := \operatorname*{args\,min}_{p \in P} \mathsf{weight}(p') \text{ in}\\
&\quad |\,P'\,|
\end{aligned}$$

Fig. 3. Use-cases for nested $\mathop{\mathcal{R}}_{p \in P} f(p)$, part 2

$$\mathrm{NWR}(s)(v) = \frac{\min_{p \in \mathsf{Paths}(s,v)} \mathsf{capacity}(p)}{\max_{p \in \mathsf{Paths}(s,v)} \mathsf{capacity}(p)}$$

Narrowest to Widest Path Ratio

$$\begin{aligned}\mathrm{LSD}(s)(v) &= \mathrm{LP}(s)(v) - \mathrm{SSSP}(s)(v)\\ &= \max_{p \in \mathsf{Paths}(s,v)} \mathsf{weight}(p) - \min_{p \in \mathsf{Paths}(s,v)} \mathsf{weight}(p)\end{aligned}$$

Difference between
Longest and Shortest Path

$$\begin{aligned}\mathrm{SP2}(s,s')(v) &= \min\left(\mathrm{SSSP}(s)(v), \mathrm{SSSP}(s')(v)\right)\\ &= \min\left(\min_{p \in \mathsf{Paths}(s,v)} \mathsf{weight}(p), \min_{p \in \mathsf{Paths}(s',v)} \mathsf{weight}(p)\right)\end{aligned}$$

Shortest Path from
Two Sources

$$\begin{aligned}\mathrm{SPR}(s,s')(v) &= \frac{\mathrm{SSSP}(s)(v)}{\mathrm{SSSP}(s')(v)}\\ &= \frac{\min_{p \in \mathsf{Paths}(s,v)} \mathsf{weight}(p)}{\max_{p \in \mathsf{Paths}(s',v)} \mathsf{weight}(p)}\end{aligned}$$

Ratio of Shortest Paths
from Two Sources

Fig. 4. Use-cases for nested $m \oplus m$

$$\mathrm{Ecc}(s) = \max_{v\in \mathsf{V}} \min_{p\in \mathsf{Paths}(s,v)} \mathsf{length}(p)$$

Eccentricity

$$\begin{aligned}\mathrm{WPG}(s) &= \min_{v\in \mathsf{V}} \mathrm{WP}(s)(v)\\ &= \min_{v\in \mathsf{V}} \max_{p\in \mathsf{Paths}(s,v)} \mathsf{capacity}(p)\end{aligned}$$

The Capacity of the Narrowest of
the Widest Paths
from s to All Vertices

$$\begin{aligned}\mathrm{LNPG}(s) &= \max_{v\in \mathsf{V}} \mathrm{LNP}(s)(v)\\ &= \mathsf{let}\ P(v) := \operatorname*{args\,min}_{p\in \mathsf{Paths}(s,v)} \mathsf{capacity}(p)\ \mathsf{in}\\ &\ \max_{v\in \mathsf{V}} \max_{p\in P(v)} \mathsf{length}(p)\end{aligned}$$

The Length of the Longest of
the Narrowest Paths
from s to All Vertices

$$\begin{aligned}\mathrm{NCC} &= \left|\bigcup_{v\in \mathsf{V}} \mathrm{CC}(v)\right|\\ &= \left|\bigcup_{v\in \mathsf{V}} \{ \min_{p\in \mathsf{Paths}(v)} \mathsf{head}(p) \}\right|\end{aligned}$$

Number of Connected Components

$$\begin{aligned}\mathrm{FRA}(s) &= \bigwedge_{v\in \mathsf{V}} \mathrm{FR}(s)(v)\\ &= \bigwedge_{v\in \mathsf{V}} \bigvee_{p\in \mathsf{Paths}(s,v)} \mathsf{True}\end{aligned}$$

Reachability to All Vertices

$$\begin{aligned}\mathrm{RFA} &= \bigcap_{s\in \mathsf{V}} \mathrm{CCS}(s)\\ &= \bigcap_{v\in \mathsf{V}} \bigcup_{p\in \mathsf{Paths}(v)} \{\, \mathsf{head}(p)\,\}\end{aligned}$$

Vertices Reachable to All Vertices

Fig. 5. Use-cases for $\underset{v\in \mathsf{V}}{\mathcal{R}}\ m$

$$\begin{aligned}\mathrm{DS}(s) &= \bigcup_{v\in \mathsf{V}\,\wedge\, \mathrm{SSSP}(s)(v) > 7} \{v\}\\ &= \bigcup_{v\in \mathsf{V}\,\wedge\, \min_{p\in \mathsf{Paths}(s,v)} \mathsf{weight}(p) > 7} \{v\}\end{aligned}$$

Vertices with the distance of at least 7

$$\mathrm{SCC}(s)(v) = \bigcup_{v'\in \mathsf{V}\,\wedge\, \bigvee_{p\in \mathsf{Paths}(v,v')} \mathsf{True}\,\wedge\, \bigvee_{p\in \mathsf{Paths}(v',v)} \mathsf{True}} \{v'\}$$

Strongly Connected Component

Fig. 6. Use-cases for $\underset{v\in \mathsf{V}\,\wedge\, m}{\mathcal{R}}\ m$

$$
\begin{aligned}
\textsc{Radius} &= \min_{s\in\{\overline{v}\}} \max_{v\in \mathrm{V}} \min_{p\in \mathsf{Paths}(s,v)} \mathsf{length}(p) && \text{Radius Sampled on vertices } \{\overline{v}\} \\
\textsc{Diam} &= \max_{s\in\{\overline{v}\}} \max_{v\in \mathrm{V}} \min_{p\in \mathsf{Paths}(s,v)} \mathsf{length}(p) && \text{Diameter Sampled on vertices } \{\overline{v}\} \\
\mathrm{DRR} &= \frac{\textsc{Diam}}{\textsc{Radius}} && \text{Diameter to Radius Ratio} \\
&= \frac{\max\limits_{s\in\{\overline{v}\}} \max\limits_{v\in \mathrm{V}} \min\limits_{p\in \mathsf{Paths}(s,v)} \mathsf{length}(p)}{\min\limits_{s\in\{\overline{v}\}} \max\limits_{v\in \mathrm{V}} \min\limits_{p\in \mathsf{Paths}(s,v)} \mathsf{length}(p)} \\
\mathrm{BC}(s) &= \mathsf{let}\ \mathsf{S} \coloneqq \lambda s, v.\ \min_{p\in \mathsf{Paths}(s,v)} \mathsf{length}(p)\ \mathsf{in} \\
&\ \mathsf{let}\ \mathsf{N} \coloneqq \lambda s, v.\ \left| \mathsf{args} \min_{p\in \mathsf{Paths}(s,v)} \mathsf{length}(p) \right|\ \mathsf{in} \\
&\ \sum_{v\in\overline{s}} \frac{\sum\limits_{\substack{v\neq t\in V\ \wedge \\ \mathsf{S}(v)(s)+\mathsf{S}(s)(t)=\mathsf{S}(v)(t)}} \mathsf{N}(v)(s) \times \mathsf{N}(s)(t)}{\sum\limits_{v\neq t\in V} \mathsf{N}(v,t)}
\end{aligned}
$$

Fig. 7. Use-cases for $r \oplus r$

BC specifies the betweenness centrality algorithm from a sampled set of nodes $\overline{s}$. For every pair of nodes (source is from sampled set and destination is over all the nodes), it calculates the number of shortest paths that goes through s. The nominator calculates the number of sortest paths (N) from v to t that passes through s. It uses a vertex-based reduction constrained by path-based reductions similar to DS. Similarly, the denominator calculates all the shortest paths from v to t. Finally, Betweenness Centrality measure is calculated using sum vertex-based reduction over sampled nodes.

$$\begin{array}{rcll}
\mathrm{NPH}(s)(v) & = & \displaystyle\sum_{p \in \mathsf{Paths}(s,v)} \mathsf{length}(p) \mapsto 1 & \text{Number of Paths Histogram} \\[2ex]
\mathrm{LSP}(s)(v) & = & \displaystyle\sum_{p' \in \underset{p \in \mathsf{Paths}(s,v)}{\operatorname{args\,min}} \mathsf{weight}(p)} \mathsf{length}(p) \mapsto 1 & \text{Length of Shortest Paths} \\[2ex]
\mathrm{CCH}(v) & = & \displaystyle\sum_{v \in \mathsf{V}} \Big(\min_{p \in \mathsf{Paths}(v)} \mathsf{head}(p) \Big) \mapsto 1 & \text{Connected Components Sizes}
\end{array}$$

Fig. 8. Use-cases with map values

2 Specification and Fusion

2.1 Semantics

SPRed

$$\left\llbracket \underset{p\in P}{\mathcal{R}}\, \mathcal{F}(p) \right\rrbracket(g) \;=\; \overline{[\mathsf{v} \mapsto \mathcal{R}\ \{\mathcal{F}(p) \mid p \in \llbracket P \rrbracket(g)(\mathsf{v})\}]}_{\mathsf{v}\in\mathsf{V}(g)}$$

SMBin

$$\llbracket m \oplus m' \rrbracket(g) \;=\; \llbracket m \rrbracket(g) \oplus \llbracket m' \rrbracket(g)$$

SMLet

$$\llbracket \mathsf{ilet}\ X \coloneqq M\ \mathsf{in}\ e \rrbracket(g) \;=\; \overline{[\mathsf{v} \mapsto \llbracket e\,[X \coloneqq \llbracket M \rrbracket(g)(\mathsf{v})] \rrbracket]}_{\mathsf{v}\in\mathsf{V}(g)}$$

Var

$$\llbracket x \rrbracket(g) \;=\; \bot$$

SVRed

$$\left\llbracket \underset{\mathsf{V}}{\mathcal{R}}\, m \right\rrbracket(g) \;=\; \mathcal{R}\ \left\{\overline{\llbracket m \rrbracket(g)(\mathsf{v})}_{\ \mathsf{v}\in\mathsf{V}(g)}\right\}$$

SRBin

$$\llbracket r \oplus r' \rrbracket(g) \;=\; \llbracket r \rrbracket(g) \oplus \llbracket r' \rrbracket(g)$$

SRLet

$$\left\llbracket \begin{array}{l} \mathsf{ilet}\ X \coloneqq M\ \mathsf{in} \\ \mathsf{mlet}\ X' \coloneqq E\ \mathsf{in} \\ \mathsf{rlet}\ X'' \coloneqq R\ \mathsf{in} \\ e \end{array} \right\rrbracket(g) \;=\; \left\llbracket e\left[X'' \coloneqq \left\llbracket R\left[X' \coloneqq \llbracket E\,[X \coloneqq \llbracket M \rrbracket(g)] \rrbracket\right] \right\rrbracket(g)\right] \right\rrbracket$$

SPaths

$$\llbracket \mathsf{Paths} \rrbracket(g) \;=\; \overline{[\mathsf{v} \mapsto \{p \mid p \in \mathsf{Paths}(g) \wedge \mathsf{tail}(p) = \mathsf{v}\}]}_{\mathsf{v}\in\mathsf{V}(g)}$$

SArgsR

$$\left\llbracket \underset{p\in P}{\mathsf{args}\,\mathcal{R}}\ \mathcal{F}(p) \right\rrbracket(g) \;=\; \overline{[\mathsf{v} \mapsto \{p \mid p \in \mathcal{P} \wedge \mathcal{F}(p) = n\}]}_{\mathsf{v}\in\mathsf{V}(g)} \quad \text{where} \quad \begin{array}{l} \mathcal{P} \;=\; \llbracket P \rrbracket(g)(\mathsf{v}) \\ \mathcal{R} \in \{\min, \max\} \\ n = \mathcal{R}\ \{\mathcal{F}(p) \mid p \in \mathcal{P}\} \end{array}$$

SMPair

$$\llbracket \langle M, M' \rangle \rrbracket(g) = \big\langle \llbracket M \rrbracket(g), \llbracket M' \rrbracket(g) \big\rangle$$

SMM

$$\llbracket \mathcal{R}\,\mathcal{F} \rrbracket = \left\llbracket \underset{p\in\mathsf{Paths}}{\mathcal{R}}\ \mathcal{F}(p) \right\rrbracket$$

SRPair

$$\llbracket \langle R, R' \rangle \rrbracket(g) = \big\langle \llbracket R \rrbracket(g), \llbracket R' \rrbracket(g) \big\rangle$$

SRR

$$\left\llbracket \mathcal{R}\ \left\langle \overline{[\mathsf{v} \mapsto n_{\mathsf{v}}]}_{\mathsf{v}\in\mathsf{V}(g)}, .., \overline{[\mathsf{v} \mapsto n'_{\mathsf{v}}]}_{\mathsf{v}\in\mathsf{V}(g)} \right\rangle \right\rrbracket = \left\llbracket \underset{\mathsf{V}}{\mathcal{R}}\ \left(\overline{[\mathsf{n} \mapsto \langle n_{\mathsf{v}},\ ..,\ n'_{\mathsf{v}} \rangle]}_{\mathsf{v}\in\mathsf{V}(g)} \right) \right\rrbracket$$

SEBin

$$\llbracket e \oplus e' \rrbracket \;=\; \llbracket e \rrbracket \oplus \llbracket e' \rrbracket$$

SEVal

$$\llbracket \mathsf{n} \rrbracket \;=\; \mathsf{n}$$

SEM

$$\llbracket d \rrbracket \;=\; d$$

SEEPair

$$\llbracket \langle E, E' \rangle \rrbracket = \big\langle \llbracket E \rrbracket, \llbracket E' \rrbracket \big\rangle$$

Fig. 9. Denotational Semantics of the language presented in Fig. 9 of the main paper. The notation $\overline{[k_i \mapsto v_i]}_i$ represents a finite map that maps each key k_i to value v_i over the range i. The notation $X \coloneqq V$ represents pointwise replacement of the variables X with the values V.

We now define a denotational semantics for the language that we presented in Fig. 9 of the main paper. We first present the semantics and then prove that it is compositional.

The semantics is defined in Fig. 9. Given a graph g, separate rules define the semantics $\llbracket\ \rrbracket$ of each term constructor. The semantics of an undefined or stuck computation is represented by $\bot$. In each rule, it is assumed that the semantics of subterms are not undefined; otherwise, the semantics of the term is undefined as well. The semantics of term constructors with no rules is $\bot$ too.

The semantics of m terms are defined by the rules SPRed, SMBin, SMLet and Var. Given a graph g, the semantic domain $\mathcal{D}_m$ of m-terms is a finite map $\mathsf{V}(g) \mapsto \mathbb{N}$ from each vertex of g to natural numbers, and $\bot$ (for undefined). The rule SPRed defines the semantics of the path-base reduction $\underset{p\in P}{\mathcal{R}}\, \mathcal{F}(p)$. (We use the notation $\overline{[k_i \mapsto v_i]}_i$ for a finite map that maps each key k_i to value v_i over the range i.) It uses the semantics of paths P that is a map from each vertex v to the set of paths to v. For each vertex v, it applies the function $\mathcal{F}$ to each path to v and then applies the reduction function $\mathcal{R}$ to the resulting values. Since the reduction functions $\mathcal{R}$ (in the semantic domain) are commutative and associative, they can be applied to the set in any order. The rule SMBin defines the the semantics of $m \oplus m'$ as the result of the operator $\oplus$ on the semantics of m and m'. Whether the notations $\mathcal{R}$ and $\oplus$ refer to the syntactic or semantic domains is clear from the context: they are in the syntactic and semantic domains when they are respectively on the left- and right-hand side of the rules. The operator $\oplus$ is simply lifted to maps of the same domain by the pointwise application for each key. The rule SMLet defines the semantics of $\mathsf{ilet}\ X := M\ \mathsf{in}\ e$ as the pointwise substitution of the variables X with the semantics of M in e. Pointwise substitution replaces variables with values from a corresponding pair of structures. (The formal definition of substitution is available in the appendix § 4.1). The rule Var states that the semantics of free variables is undefined.

The semantics of r terms is defined by the rules SVRed, SRBin, and SRLet and Var. The domain D_r of of r-terms is the natural numbers $\mathbb{N}$ and $\bot$. The rule SVRed defines the semantics of the vertex-based reduction $\underset{\mathsf{V}}{\mathcal{R}}\ m$ using the map resulted from the semantics of m; it reduces the values of the map for all vertices. The rule SRBin defines the semantics of $r \oplus r'$ as the result of applying the operator $\oplus$ to the semantics of r and r'. The rule SRLet defines the semantics of triple-let terms by three subsequent substitutions: the substitution of the variables X with the semantics of M in E, the substitution of the variables X' with the semantics of E in R, and finally the substitution of the variables X'' with the semantics of R in e.

The semantics of paths P is defined by the rules SPaths and SArgsR. The rule SPaths defines the semantics of the term Paths as a map from each vertex to the set of paths to the vertex. The rule SArgsR defines the semantics of $\underset{p \in P}{\mathrm{args}\,\mathcal{R}}\ \mathcal{F}(p)$ where $\mathcal{R}$ is min or max using the map resulted from the semantics $[\![\, P \,]\!]$ of P; it maps each vertex v to a subset of the paths that $[\![\, P \,]\!]$ maps v to: the paths that their $\mathcal{F}$ value is the minimum or the maximum.

The rules SMPair, SRPair, and SEEPair define the semantics of pairs of M, R and E inductively. The two rules SMM and SRR reduce the semantics of single factored reductions to normal reductions. The rule SMM defines the semantics of $\mathcal{R}\ \mathcal{F}$ as a path-based reduction on the paths Paths. The rule SRR defines the semantics of $\mathcal{R}\ \left\langle \overline{[\mathsf{v} \mapsto n_{\mathsf{v}}]}_{\mathsf{v} \in \mathsf{V}(g)}, .., \overline{[\mathsf{v} \mapsto n'_{\mathsf{v}}]}_{\mathsf{v} \in \mathsf{V}(g)} \right\rangle$ as a vertex-based reduction on $\overline{\langle n_{\mathsf{v}}, .., n'_{\mathsf{v}} \rangle}_{\mathsf{v} \in \mathsf{V}(g)}$. The rules SEBin, SEVal, and SEM define the semantics of expressions e. An expression e can represent both a number and a vertex-based reduction. The operator $\oplus$ is overloaded for both numbers and maps in the semantic domain.

The semantics is compositional. If two terms are semantically equivalent, replacing one with the other in any context is semantics-preserving. Compositionality of the semantics is used to prove that the fusion transformations are semantic-preserving. The following theorem states that all the terms r, m, M and R are compositional. The proofs are available in the appendix § 4.2.

Lemma 1 (Compositionality).
For all r, r' *and* $\mathbb{R}$, *if* $[\![\, r \,]\!] = [\![\, r' \,]\!]$ *then* $[\![\, \mathbb{R}[r] \,]\!] = [\![\, \mathbb{R}[r'] \,]\!]$.
For all m, m', *and* $\mathbb{M}$, *if* $[\![\, m \,]\!] = [\![\, m' \,]\!]$ *then* $[\![\, \mathbb{M}[m] \,]\!] = [\![\, \mathbb{M}[m'] \,]\!]$.
For all M, M', *and* $\mathbb{M}\mathrm{s}$, *if* $[\![\, M \,]\!] = [\![\, M' \,]\!]$ *then* $[\![\, \mathbb{M}\mathrm{s}[M] \,]\!] = [\![\, \mathbb{M}\mathrm{s}[M'] \,]\!]$.
For all R, R', *and* $\mathbb{R}\mathrm{s}$, *if* $[\![\, R \,]\!] = [\![\, R' \,]\!]$ *then* $[\![\, \mathbb{R}\mathrm{s}[R] \,]\!] = [\![\, \mathbb{R}\mathrm{s}[R'] \,]\!]$.

2.2 Language and Fusion Extensions

In this section, we describe the language extensions and their corresponding fusion rules. Fig. 10 represents the extensions to the syntax for the following subsections.

$$
\begin{array}{lcll}
r & := & \boxed{\underset{\mathsf{V}}{\mathcal{R}}\, m \oplus r} \mid \boxed{\boxed{\underset{v\in\mathsf{V}}{\mathcal{R}}\, m}} \mid r \oplus r \mid \dbox{$\circ\, r \mid n$} \mid & \text{Vertex-based Reduction} \\
 & & \mathsf{ilet}\ X := M\ \mathsf{in} & \\
 & & \mathsf{mlet}\ X := E\ \mathsf{in} & \\
 & & \mathsf{rlet}\ X := R\ \mathsf{in} & \\
 & & e \mid x & \\
m & := & \underset{p\in P}{\mathcal{R}}\, F(p) \mid m \oplus m \mid & \text{Path-based Reduction} \\
 & & \boxed{\boxed{\mathsf{ilet}\ X := M\ \mathsf{in}\ v\ e}} \mid x & \\
P & := & \boxed{\boxed{\mathsf{Paths}(v) \mid \mathsf{Paths}(v, v')}} \mid \mathsf{args}\underset{p\in P}{\mathcal{R}}\, F(p) & \text{Paths} \\
M & := & \langle M, M\rangle \mid \boxed{\boxed{\underset{c}{\mathcal{R}}\, \mathcal{F}}} & \\
\mathbb{R} & := & \ldots \mid \boxed{\underset{\mathsf{V}}{\mathcal{R}}\, m \oplus \mathbb{R}} & \text{Context for } r \\
v & & & \text{Vertex Variable} \\
s & := & v \mid \bot & \text{Source} \\
o & := & \rightarrow \mid \leftarrow & \text{Orientation} \\
c & := & s\ o \mid \langle c, c\rangle & \text{Path Configuration} \\
\mathcal{R} & := & \ldots \mid \dbox{$\cup \mid \cap$} & \text{Reduction Operation} \\
\mathcal{F} & := & \ldots \mid \dbox{$\mathsf{head} \mid \mathsf{penultimate}$} & \text{Path Function}
\end{array}
$$

Fig. 10. Extended Syntax. Dashed boxes for § 2.2.2 and § 2.2.3, solid boxes for § 2.2.6, and double solid boxes for § 2.2.4

2.2.1 Common Operation Elimination

IElim

$$\left(\begin{array}{l} \text{ilet } \langle X_1, X_2\rangle := \langle \mathcal{R}\,\mathcal{F}, \mathcal{R}\,\mathcal{F}\rangle \text{ in} \\ \text{mlet } X' := E \text{ in} \\ \text{rlet } X'' := R \text{ in} \\ e \end{array}\right) \Rightarrow \left(\begin{array}{l} \text{ilet } X_1 := \mathcal{R}\,\mathcal{F} \text{ in} \\ \text{mlet } X' := E[X_2 \mapsto X_1] \text{ in} \\ \text{rlet } X'' := R \text{ in} \\ e \end{array}\right)$$

ICom

$$\left(\begin{array}{l} \text{ilet } \langle X_1, X_2\rangle := \langle M_1, M_2\rangle \text{ in} \\ \text{mlet } X' := E \text{ in} \\ \text{rlet } X'' := R \text{ in} \\ e \end{array}\right) \Rightarrow \left(\begin{array}{l} \text{ilet } \langle X_2, X_1\rangle := \langle M_2, M_1\rangle \text{ in} \\ \text{mlet } X' := E \text{ in} \\ \text{rlet } X'' := R \text{ in} \\ e \end{array}\right)$$

IAssL

$$\left(\begin{array}{l} \text{ilet } \langle X_1, \langle X_2, X_3\rangle\rangle := \langle M_1, \langle M_2, M_3\rangle\rangle \text{ in} \\ \text{mlet } X' := E \text{ in} \\ \text{rlet } X'' := R \text{ in} \\ e \end{array}\right) \Rightarrow \left(\begin{array}{l} \text{ilet } \langle\langle X_1, X_2\rangle, X_3\rangle := \langle\langle X_1, X_2\rangle, M_3\rangle \text{ in} \\ \text{mlet } X' := E \text{ in} \\ \text{rlet } X'' := R \text{ in} \\ e \end{array}\right)$$

IAssR

$$\left(\begin{array}{l} \text{ilet } \langle\langle X_1, X_2\rangle, X_3\rangle := \langle\langle M_1, M_2\rangle, M_3\rangle \text{ in} \\ \text{mlet } X' := E \text{ in} \\ \text{rlet } X'' := R \text{ in} \\ e \end{array}\right) \Rightarrow \left(\begin{array}{l} \text{ilet } \langle X_1, \langle X_2, X_3\rangle\rangle := \langle M_1, \langle M_2, M_3\rangle\rangle \text{ in} \\ \text{mlet } X' := E \text{ in} \\ \text{rlet } X'' := R \text{ in} \\ e \end{array}\right)$$

Fig. 11. Common Operation Elimination

Fusion factors the path-based reduction, and vertex-based mappings and reductions. The factoring facilitates common operation elimination. For example, if a path-based reduction is calculated twice and assigned to two sets of variables, the extra calculation can be eliminated and the result of one calculation can be assigned to both sets of variables.

Fig. 11 shows the elimination rules for path-based reductions. The rule IElim applies to adjacent similar path-based reductions. The second reduction is eliminated. The variables for the second reduction are substituted with the variables for the first reduction. To bring two path-based reductions adjacent to each other, the rules ICom, IAssL and IAssR state the commutativity and associativity properties of pairs of path-based reductions.

Similar eliminations can be applied to the factored vertex-based mappings in the second let and the factored vertex-based reductions in the third let.

As an example of common operation elimination, see the fusion of the use-case DRR in § 2.3.

2.2.2 Domain

The scalar semantic domain of the core language was confined to the natural numbers. The domain can be simply extended to booleans, vertex identifiers and also sets of values. The reduction operations are extended with union ∪ and intersection ∩ and the path functions are extended with head and penultimate. The function head returns the identifier of the head vertex of the path and the function penultimate returns the identifier of the penultimate (that is the vertex before the last) of the path. These extensions are shown in dashed boxes in Fig. 10

2.2.3 Unary operations and Literals

$$\textsc{FILit} \quad n \;\Rightarrow\; \mathsf{ilet}\ x := \bot\ \mathsf{in}\ n \qquad\qquad \textsc{FMLVar} \quad x \;\Rightarrow\; \mathsf{ilet}\ x' := \bot\ \mathsf{in}\ x$$

$$\textsc{FMPair}' \quad \langle X, x\rangle := \langle M, \bot\rangle \;\rightarrow\; X := M \qquad\qquad \textsc{FMPair}'' \quad \langle x, X\rangle := \langle \bot, M\rangle \;\rightarrow\; X := M$$

$$\textsc{FRLit} \quad n \;\Rightarrow\; \begin{array}{l} \mathsf{ilet}\ x := \bot\ \mathsf{in} \\ \mathsf{mlet}\ x' := \bot\ \mathsf{in} \\ \mathsf{rlet}\ x'' := \bot\ \mathsf{in}\ n \end{array}$$

$$\textsc{FRPair}' \quad \langle X, x\rangle := \langle R, \bot\rangle \;\rightarrow\; X := R \qquad\qquad \textsc{FRPair}'' \quad \langle x, X\rangle := \langle \bot, R\rangle \;\rightarrow\; X := R$$

$$\textsc{FIUni} \quad \circ\,(\mathsf{ilet}\ X := M\ \mathsf{in}\ e) \;\Rightarrow\; \mathsf{ilet}\ X := M\ \mathsf{in} \circ e \qquad\qquad \textsc{FRUni} \quad \circ \left(\begin{array}{l} \mathsf{ilet}\ X := M\ \mathsf{in} \\ \mathsf{mlet}\ X' := E\ \mathsf{in} \\ \mathsf{rlet}\ X'' := R\ \mathsf{in} \\ e \end{array}\right) \;\Rightarrow\; \left(\begin{array}{l} \mathsf{ilet}\ X := M\ \mathsf{in} \\ \mathsf{mlet}\ X' := E\ \mathsf{in} \\ \mathsf{rlet}\ X'' := R\ \mathsf{in} \\ \circ\, e \end{array}\right)$$

Fig. 12. Extended Fusion Rules for Unary operators and constants

In this section, we present the fusion rules for the natural number literals n and unary operators ∘. As other rules expect terms to be in the let form, the two rules FILit and FRLit transform a literal to dummy *m* let and *r* let forms. Since the two rules FMPair and FRPair apply to only non-⊥ reductions, the rules FMPair′, FMPair″, FRPair′ and FRPair″ remove the dummy ⊥ reductions. The two rules FIUni and FRUni simply apply the unary operator ∘ to the resulting expression *e*.

2.2.4 *Vertex Variables*

$$\textsc{FPRed}\quad \underset{p \in \mathsf{Paths}(v)}{\mathcal{R}} \mathcal{F}(p) \Rightarrow_m \mathsf{ilet}\ x \coloneqq \underset{\bot \rightarrow}{\mathcal{R}} \mathcal{F}\ \mathsf{in}\ v\ x$$

$$\textsc{FPRed}'\quad \underset{p \in \mathsf{Paths}(v,v')}{\mathcal{R}} F(p) \Rightarrow_m \mathsf{ilet}\ x \coloneqq \underset{v \rightarrow}{\mathcal{R}} \mathcal{F}\ \mathsf{in}\ v'\ x$$

$$\textsc{FPRed}''\quad \underset{p \in \mathsf{Paths}(v,v')}{\mathcal{R}} \mathcal{F}(p) \Rightarrow_m \mathsf{ilet}\ x \coloneqq \underset{v' \leftarrow}{\mathcal{R}} \mathcal{F}\ \mathsf{in}\ v\ x$$

$$\textsc{FILetBin}\quad \begin{array}{c}(\mathsf{ilet}\ X_1 \coloneqq M_1\ \mathsf{in}\ v\ e_1) \oplus (\mathsf{ilet}\ X_2 \coloneqq M_2\ \mathsf{in}\ v\ e_2) \\ \Rightarrow_m \\ \mathsf{ilet}\ \langle X_1, X_2 \rangle \coloneqq \langle M_1, M_2 \rangle\ \mathsf{in}\ v\ (e_1 \oplus e_2)\end{array} \quad \text{if} \quad \begin{array}{l}\mathsf{free}(e_1) \cap X_2 = \emptyset \\ \mathsf{free}(e_2) \cap X_1 = \emptyset\end{array}$$

$$\textsc{FMPair}\quad \left\langle \underset{c}{\mathcal{R}} \mathcal{F}, \underset{c'}{\mathcal{R}'} \mathcal{F}' \right\rangle \Rightarrow_M \underset{\langle c,c' \rangle}{\mathcal{R}''} \mathcal{F}''$$
$$\text{where}\quad \begin{array}{l} f'' \coloneqq \lambda p.\ \langle F'(p), F(p) \rangle \\ \mathcal{R}''(\langle a, b \rangle, \langle a', b' \rangle) \coloneqq \\ \qquad \langle \mathcal{R}(a, a'), \mathcal{R}'(b, b') \rangle \end{array}$$

$$\textsc{FVRed}\quad \underset{v \in \mathsf{V}}{\mathcal{R}} (\mathsf{ilet}\ X \coloneqq \underset{c}{\mathcal{R}'} f\ \mathsf{in}\ v\ e) \Rightarrow_r \begin{array}{l} \mathsf{ilet}\ X \coloneqq \underset{c}{\mathcal{R}'} f\ \mathsf{in} \\ \mathsf{mlet}\ x \coloneqq e\ \mathsf{in} \\ \mathsf{rlet}\ x' \coloneqq \mathcal{R}\ x\ \mathsf{in}\ x' \end{array}$$

Fig. 13. Extended Fusion Rules for Vertex Variables

The syntax of the core language offers the simple term Paths that does not specify the source and destination of paths. Further, the vertex-based reduction $\underset{\mathsf{V}}{\mathcal{R}}\ m$ does not bind a vertex variable. In this section, we extend the core syntax with path terms that can specify vertex variables as source and destination and vertex-based reductions that can bind vertex variables. We extend the fusion rules for the extended syntax.

In Fig. 10, the double boxes shows the extension to the core syntax presented in Fig. 9 to support vertex variables. Only the changed or new non-terminals are shown and the updated parts are boxed with solid lines. The extended vertex-based reduction $\underset{v \in \mathsf{V}}{\mathcal{R}}\ m$ binds the vertex variable v. The path constructors specify source and destination: the term $\mathsf{Paths}(v)$ specifies the set of paths with any source and the destination v and the term $\mathsf{Paths}(v, v')$ specifies the set of paths with the source v and the destination v'. In its simplest form, a factored path-based reduction M calculates the reduction over paths from a source vertex v to every destination vertex v' and stores the result in the destination vertices v'. It can also calculate the reduction over paths from every source vertex v to a destination vertex v' and store the result in the source vertices v. We call the vertex variable where the result is stored, the target vertex. The let constructor $\mathsf{ilet}\ X \coloneqq M\ \mathsf{in}\ v\ e$ of the path-based reductions m carries the vertex v that stores the result of the factored reduction M with the expression e.

The source s of paths can be either a vertex v or none $\bot$. The orientation o of paths is either forward $\longrightarrow$ or backward $\longleftarrow$. The configuration c of paths is the pair of their source and orientation, or a pair of other configurations. A single factored path-based reduction $\underset{c}{\mathcal{R}} \mathcal{F}$ carries its configuration c.

Fig. 13 shows the extension of the core fusion rules presented in Fig. 11. Only the updated fusion rules are shown. The rules FPRed, FPRed′ and FPRed″ convert path-based reductions over paths terms to the let form. The rule FPRed converts a path-based reduction over $\mathsf{Paths}(v)$ to a let term with a factored path-based reduction that has no source $\bot$, forward orientation $\longrightarrow$, and the target vertex v. The rules FPRed′ and FPRed″ both convert a path-based reduction over $\mathsf{Paths}(v, v')$ to let forms. The former stores the results in the destination vertices and the latter stores the results in the source vertices. The former results in a let term with with a factored path-based reduction that has source v, forward orientation $\rightarrow$, and the target vertex v'. The latter, on the other hand, results in a let term with a factored path-based reduction that has source v', backward orientation $\leftarrow$, and the target vertex v.

The rule FILetBin fuses an operation between two path-based reductions in the let form to one. The operation can be applied to the resulting expressions of the two let terms only if they are stored in the same target vertex. Therefore, the rule checks that the explicit target vertex of the two let terms match.

The rule FMPair simply passes the configurations of the two reductions to the fused reduction.

A vertex-based reduction applies a reduction to the results of a path-based reduction over all vertices. The rule FVRed converts the application of a vertex-based reduction to a path-based reduction to the triple-let form; it checks that the vertex bound by the nesting vertex-based reduction matches the target vertex of the path-based reduction.

2.2.5 Syntactic Sugar

FMRed

$$\mathcal{F}\,(\arg \underset{p\in P}{\mathcal{R}}\ \mathcal{F}'(p)) \quad := \quad \text{ilet } \langle x, x'\rangle := \underset{p\in P}{\mathcal{R}'}\ \mathcal{F}''(p) \text{ in } x' \qquad \text{where} \qquad \mathcal{R} \in \{\min, \max\}$$

$$\mathcal{F}'' \ := \ \lambda p.\ \langle \mathcal{F}'(p), \mathcal{F}(p)\rangle$$

$$\mathcal{R}'(\langle a, b\rangle, \langle a', b'\rangle) \ := \ \text{if } (\mathcal{R}(a, a') = a) \text{ then } \langle a, b\rangle \text{ else } \langle a', b'\rangle$$

PSize

$$|P| \quad := \quad \sum_{p\in P} 1$$

ROp

$$\underset{v\in\{v_1,..,v_n\}}{\mathcal{R}} m \quad := \quad ((m[v := v_1]\ \mathcal{R}\ m[v := v_2])\ \mathcal{R}\ ..\ m[v := v_n])$$

VSel

$$\underset{v\in \mathsf{V} \wedge m'}{\mathcal{R}} m \quad := \quad \begin{array}{l} \text{ilet } \langle x, x'\rangle := \underset{v\in \mathsf{V}}{\mathcal{R}'} \langle m', m\rangle \text{ in} \\ \text{if } x \text{ then } x' \text{else } \bot \end{array}$$

where

$$\begin{array}{l} \mathcal{R}'(\langle a, b\rangle, \langle a', b'\rangle) := \\ \quad \text{if } (a \wedge a') \text{ then } \langle a, \mathcal{R}(b, b')\rangle \\ \quad \text{else if } (a') \text{ then } \langle a', b'\rangle \\ \quad \text{else } \langle a, b\rangle \end{array}$$

Fig. 14. Syntactic Sugar

Syntactic sugar enable concise specifications. In Fig. 14, we present the syntactic sugar and the rules that desugar them.

The term $\mathcal{F}(\arg \underset{p\in P}{\mathcal{R}}\ \mathcal{F}'(p))$ where $\mathcal{R}$ is either min or max first finds a path p in P with the minimum or maximum value for the function $\mathcal{F}'$ and then returns the result of applying $\mathcal{F}$ to p. It is used to specify the BFS use-case. The rule FMRed expands this term to a path-based reduction in the let form ilet $\langle x, x'\rangle := \underset{p\in P}{\mathcal{R}'}\ \mathcal{F}''(p)$ in x'. The path function $\mathcal{F}''$ returns the pair of the results of $\mathcal{F}'$ and $\mathcal{F}$. The reduction function $\mathcal{R}'$ returns the input pair with the minimum or maximum first element.

The term $|P|$ specifies the size of the set of paths P. It is used to specify the NSP use-case. The rule PSize simply expands it to the path-based reduction $\sum_{p\in P} 1$ that counts the number of paths.

The term $\underset{v\in\{v_1,..,v_n\}}{\mathcal{R}} m$ is a vertex-based reduction over a limited set of vertices $\{v_1, .., v_n\}$. It is used to specify the Radius use-case. The rule ROp expands this term to operations between to path-based reductions $m[v := v_i], i \in \{1..n\}$. The operation corresponds to the reduction function $\mathcal{R}$; for example, the reduction function $\sum$ is unrolled to the operation $+$.

The term $\underset{v\in \mathsf{V} \wedge m'}{\mathcal{R}} m$ specifies a vertex-based reduction of m over the selected vertices v for which m' evaluates to true. This idiom was used to specify the DS use-case. The rule VSel expands it to the path-based reduction $\underset{v\in \mathsf{V}}{\mathcal{R}'} \langle m', m\rangle$. The path-based reduction calculates a pair of values for m' and m at every vertex. Then, the vertex-based reduction $\mathcal{R}'$ only reduces the second elements of the pairs whose first element is true. Given two input pair, the vertex-based reduction $\mathcal{R}'$ applies the reduction $\mathcal{R}$ to the second elements if the first elements of both pairs are true. Otherwise, the pair whose first element is true is selected. If the first element of both pairs is false, either of them can be selected; this definition selects the first. Finally, in the following if expression, there are two cases. If there has been pairs whose first element is true, the result of the reduction is a pair with true as the first element and the result of the reduction as the second element. In this case, the second element is returned. Otherwise, there has not been any pair with true as the first element. In this case, none is returned.

2.2.6 Nested Triple-lets

$$\text{FRR}\quad \frac{r_1 \quad \Rightarrow_r \quad r_2}{\mathbb{R}[r_1] \quad \Rightarrow_r \quad \mathbb{R}[r_2]}$$

Fig. 15. Extended Fusion Rules for Multiple Rounds

The core syntax supports expressions that can be fused to a single iteration-map-reduce triple-let term. In this subsection, we extend the core syntax to support nested vertex-based reductions, and extend the fusion rules to fuse nested reductions. Nested triple-let terms that are closed (i.e. do not have free variables) can be factored out. Thus, nested triple-let terms can be translated to a sequence of iteration-map-reduce rounds on the graph.

In Fig. 10, the single boxes show the extensions to the core syntax presented in Fig. 9 to support multiple rounds. The constructors of vertex-based reductions r include the new term $\underset{\mathsf{V}}{\mathcal{R}}\ m \oplus r$ where an operation $\oplus$ can be applied to a path-based reduction m and a nested vertex-based reduction r. This nested r leads to a round of iteration-map-reduce. Similarly, the vertex-based reduction contexts $\mathbb{R}$ include the term $\underset{\mathsf{V}}{\mathcal{R}}\ m \oplus \mathbb{R}$ so that the nested vertex-based reductions can be fused as well. As Fig. 15 shows, the fusion rules are extended by the rule FRR to allow the fusion of nested vertex-based reductions.

For example, consider the following use-case LTrust that calculates the capacity of narrowest path to the nodes that fall out of the radius from the node s.

$$\begin{aligned}
\text{LT{\scriptsize RUST}}(s) \quad = \quad & \mathsf{let}\ \text{SSSP} := \lambda s, v.\ \min_{p \in \mathsf{Paths}(s,v)} \mathsf{weight}(p)\ \mathsf{in} \\
& \mathsf{let}\ \text{NP} := \lambda s, v.\ \min_{p \in \mathsf{Paths}(s,v)} \mathsf{capacity}(p)\ \mathsf{in} \\
& \min_{v \in \mathsf{V}\,\wedge\, \text{SSSP}(s,v)\, <\, \text{R{\scriptsize ADIUS}}} \text{NP}(s, v)
\end{aligned}$$

Unrolling the let terms results in the following:

$$\text{LT{\scriptsize RUST}}(s) \quad = \quad \min_{v \in \mathsf{V}\,\wedge\, \left(\min\limits_{p \in \mathsf{Paths}(s,v)} \mathsf{weight}(p)\right) < \text{R{\scriptsize ADIUS}}} \left(\min_{p \in \mathsf{Paths}(s,v)} \mathsf{capacity}(p)\right)$$

By the rule VSel, this specification is desugared to the following:

$$\text{LT{\scriptsize RUST}}(s) \quad = \quad \underset{v \in \mathsf{V}}{\mathcal{R}} \left\langle \left(\min_{p \in \mathsf{Paths}(s,v)} \mathsf{weight}(p)\right) < \text{R{\scriptsize ADIUS}}, \left(\min_{p \in \mathsf{Paths}(s,v)} \mathsf{capacity}(p)\right) \right\rangle$$

$$\begin{aligned}
\text{where} \quad & \mathcal{R}(b, \langle a', b' \rangle) := \\
& \quad \mathsf{if}\ (a')\ \mathsf{then}\ \min(b, b') \\
& \quad \mathsf{else}\ b
\end{aligned}$$

We note that in the above specification, the path-based reduction $\text{SSSP}(s, v) < \text{R{\scriptsize ADIUS}}$ includes the nested vertex-based reduction Radius. From Fig. 2, Radius can be fused to following:

$$\textsc{Radius} \;=\; \left(\begin{array}{l} \mathsf{ilet}\ \langle x, y\rangle \;:=\; \underset{\langle s_1,s_2\rangle}{\mathcal{R}'}\ \mathcal{F}\ \mathsf{in} \\ \mathsf{mlet}\ \langle x', y'\rangle \;:=\; \langle x, y\rangle\ \mathsf{in} \\ \mathsf{rlet}\ \langle x'', y''\rangle \;:=\; \mathcal{R}''\ \langle x', y'\rangle\ \mathsf{in} \\ \min(x'', y'') \end{array}\right) \quad \text{where} \quad \begin{array}{l} \mathcal{F} := \lambda p.\ \langle \mathsf{length}(p), \mathsf{length}(p)\rangle \\ \mathcal{R}'\ (\langle a, b\rangle, \langle a', b'\rangle) := \\ \qquad \langle \min(a, a'), \min(b, b')\rangle \\ \mathcal{R}''\ (\langle a, b\rangle, \langle a', b'\rangle) := \\ \qquad \langle \max(a, a'), \max(b, b')\rangle \end{array}$$

Therefore, The rules FRR can be used to fuse the nested Radius term to the above triple-let term. Then, since Radius is a closed term, it can be factored out as a let term. Thus, LTrust can be rewritten as follows:

$$\begin{array}{rcl} \textsc{LTrust}(s) & = & \mathsf{let}\ \mathit{radius} := \left(\begin{array}{l} \mathsf{ilet}\ \langle x, y\rangle \;:=\; \underset{\langle s_1,s_2\rangle}{\mathcal{R}'}\ \mathcal{F}\ \mathsf{in} \\ \mathsf{mlet}\ \langle x', y'\rangle \;:=\; \langle x, y\rangle\ \mathsf{in} \\ \mathsf{rlet}\ \langle x'', y''\rangle \;:=\; \mathcal{R}''\ \langle x', y'\rangle\ \mathsf{in} \\ \min(x'', y'') \end{array}\right)\ \mathsf{in} \\ & & \underset{v \in \mathsf{V}}{\mathcal{R}} \left\langle \left(\underset{p \in \mathsf{Paths}(s,v)}{\min} \mathsf{weight}(p)\right) < \mathit{radius}, \left(\underset{p \in \mathsf{Paths}(s,v)}{\min} \mathsf{capacity}(p)\right)\right\rangle \end{array}$$

By the rule FPRed (and then for the first element of the pair, the rules FMLVar, FILetBin and FMPair′), it can be fused to the following:

$$\begin{array}{rcl} \textsc{LTrust}(s) & = & \mathsf{let}\ \mathit{radius} := \left(\begin{array}{l} \mathsf{ilet}\ \langle x, y\rangle \;:=\; \underset{\langle s_1,s_2\rangle}{\mathcal{R}'}\ \mathcal{F}\ \mathsf{in} \\ \mathsf{mlet}\ \langle x', y'\rangle \;:=\; \langle x, y\rangle\ \mathsf{in} \\ \mathsf{rlet}\ \langle x'', y''\rangle \;:=\; \mathcal{R}''\ \langle x', y'\rangle\ \mathsf{in} \\ \min(x'', y'') \end{array}\right)\ \mathsf{in} \\ & & \underset{v \in \mathsf{V}}{\mathcal{R}} \left\langle \mathsf{ilet}\ x := \underset{s}{\min}\ \mathsf{weight}\ \mathsf{in}\ x < \mathit{radius},\ \ \mathsf{ilet}\ y := \underset{s}{\min}\ \mathsf{capacity}\ \mathsf{in}\ y\right\rangle \end{array}$$

By the rule FILetBin, it is fused to the following:

$$\begin{array}{rcl} \textsc{LTrust}(s) & = & \mathsf{let}\ \mathit{radius} := \left(\begin{array}{l} \mathsf{ilet}\ \langle x, y\rangle \;:=\; \underset{\langle s_1,s_2\rangle}{\mathcal{R}'}\ \mathcal{F}\ \mathsf{in} \\ \mathsf{mlet}\ \langle x', y'\rangle \;:=\; \langle x, y\rangle\ \mathsf{in} \\ \mathsf{rlet}\ \langle x'', y''\rangle \;:=\; \mathcal{R}''\ \langle x', y'\rangle\ \mathsf{in} \\ \min(x'', y'') \end{array}\right)\ \mathsf{in} \\ & & \underset{v \in \mathsf{V}}{\mathcal{R}} \left(\mathsf{ilet}\ \langle x, y\rangle \;:=\; \langle \underset{s}{\min}\ \mathsf{weight}, \underset{s}{\min}\ \mathsf{capacity}\rangle\ \mathsf{in}\ \langle x < \mathit{radius}, y\rangle\right) \end{array}$$

By the rule FMPair, it is fused to the following:

$$\begin{array}{rcl} \textsc{LTrust}(s) & = & \mathsf{let}\ \mathit{radius} := \left(\begin{array}{l} \mathsf{ilet}\ \langle x, y\rangle \;:=\; \underset{\langle s_1,s_2\rangle}{\mathcal{R}'}\ \mathcal{F}\ \mathsf{in} \\ \mathsf{mlet}\ \langle x', y'\rangle \;:=\; \langle x, y\rangle\ \mathsf{in} \\ \mathsf{rlet}\ \langle x'', y''\rangle \;:=\; \mathcal{R}''\ \langle x', y'\rangle\ \mathsf{in} \\ \min(x'', y'') \end{array}\right)\ \mathsf{in} \\ & & \underset{v \in \mathsf{V}}{\mathcal{R}} \left(\mathsf{ilet}\ \langle x, y\rangle \;:=\; \underset{\langle s,s\rangle}{\mathcal{R}'''}\ \mathcal{F}'\ \mathsf{in}\ \langle x < \mathit{radius}, y\rangle\right) \\ & & \\ & & \text{where}\ \ \begin{array}{l} \mathcal{F}' := \lambda p.\ \langle \mathsf{weight}(p), \mathsf{capacity}'(p)\rangle \\ \mathcal{R}'''(\langle a, b\rangle, \langle a', b'\rangle) := \\ \qquad \langle \min(a, a'), \min(b, b')\rangle \end{array} \end{array}$$

By the rule FVRed, it is fused to the following:

$$
\begin{aligned}
\textsc{LTrust}(s) \quad = \quad & \mathsf{let}\ \mathit{radius} := \left(\begin{array}{l}
\mathsf{ilet}\ \langle x, y\rangle := \underset{\langle s_1,s_2\rangle}{\mathcal{R}'}\ \mathcal{F}\ \mathsf{in} \\
\mathsf{mlet}\ \langle x', y'\rangle := \langle x, y\rangle\ \mathsf{in} \\
\mathsf{rlet}\ \langle x'', y''\rangle := \mathcal{R}''\ \langle x', y'\rangle\ \mathsf{in} \\
\min(x'', y'')
\end{array}\right)\ \mathsf{in} \\
& \left(\begin{array}{l}
\mathsf{ilet}\ \langle x, y\rangle := \underset{\langle s,s\rangle}{\mathcal{R}''}\ \mathcal{F}' \\
\mathsf{mlet}\ \langle x', y'\rangle :=\ \mathsf{in}\ \langle x < \mathit{radius}, y\rangle \\
\mathsf{rlet}\ x'' := \mathcal{R}\ \langle x', y'\rangle\ \mathsf{in} \\
x''
\end{array}\right)
\end{aligned}
$$

The above specification is the sequence of two iteration-map-reduce triple let terms.

2.3 Example Fusions

We saw the fusion of the Radius use-case in the paper, Fig. 2, and the fusion of the LTrust use-case in § 2.2.6. In this subsection, we present the fusion of the DS and DRR use-cases.

$$\begin{aligned}
&\mathrm{DS}(s) \\
&= \bigcup_{v\in\mathsf{V}\,\wedge\,\left(\min\limits_{p\in\mathsf{Paths}(s,v)} \mathsf{weight}(p)\right) > 7} \{v\} && \text{By VSel} \\
&= \mathop{\mathcal{R}}_{v\in\mathsf{V}} \left\langle \left(\min_{p\in\mathsf{Paths}(s,v)} \mathsf{weight}(p)\right) > 7, \{v\} \right\rangle \quad \text{where} \quad \begin{array}{l} \mathcal{R}(\langle a, b\rangle, \langle a', b'\rangle) := \\ \quad \mathsf{if}\ (a \wedge a')\ \mathsf{then}\ \langle a, b \cup b'\rangle \\ \quad \mathsf{else}(a)\ \mathsf{then}\ \langle a, b\rangle \\ \quad \mathsf{else}\ \langle a', b'\rangle \end{array} && \text{By FPRed and FILit} \\
&= \mathop{\mathcal{R}}_{v\in\mathsf{V}} \langle \left(\mathsf{ilet}\ x := \underset{s}{\mathsf{min}}\ \mathsf{weight}\ \mathsf{in}\ x\right) > \mathsf{ilet}\ x' := \bot\ \mathsf{in}\ 7, \\
&\qquad\quad \mathsf{ilet}\ x'' := \bot\ \mathsf{in}\ \{v\}\rangle \quad \text{where} \quad \begin{array}{l} \mathcal{R}(\langle a, b\rangle, \langle a', b'\rangle) := \\ \quad \mathsf{if}\ (a \wedge a')\ \mathsf{then}\ \langle a, b \cup b'\rangle \\ \quad \mathsf{else}(a)\ \mathsf{then}\ \langle a, b\rangle \\ \quad \mathsf{else}\ \langle a', b'\rangle \end{array} && \text{By FILetBin} \\
&= \mathop{\mathcal{R}}_{v\in\mathsf{V}} \left(\mathsf{ilet}\ \langle\langle x, x'\rangle, x''\rangle := \langle\langle \underset{s}{\mathsf{min}}\ \mathsf{weight}, \bot\rangle, \bot\rangle\ \mathsf{in}\ \langle x > 7, \{v\}\rangle\right) && \text{By FMPair}' \\
&= \mathop{\mathcal{R}}_{v\in\mathsf{V}} \left(\mathsf{ilet}\ x := \underset{s}{\mathsf{min}}\ \mathsf{weight}\ \mathsf{in}\ \langle x > 7, \{v\}\rangle\right) && \text{By FVRed} \\
&= \left(\begin{array}{l} \mathsf{ilet}\ x := \underset{s}{\mathsf{min}}\ \mathsf{weight}\ \mathsf{in} \\ \mathsf{mlet}\ x' := \langle x > 7, \{v\}\rangle\ \mathsf{in} \\ \mathsf{rlet}\ x'' := \mathcal{R}\ x'\ \mathsf{in} \\ x'' \end{array}\right) \quad \text{where} \quad \begin{array}{l} \mathcal{R}(\langle a, b\rangle, \langle a', b'\rangle) := \\ \quad \mathsf{if}\ (a \wedge a')\ \mathsf{then}\ \langle a, b \cup b'\rangle \\ \quad \mathsf{else}(a)\ \mathsf{then}\ \langle a, b\rangle \\ \quad \mathsf{else}\ \langle a', b'\rangle \end{array}
\end{aligned}$$

$$
\begin{aligned}
\mathrm{DRR} &= \frac{\textsc{Diam}}{\textsc{Radius}} \\
&= \frac{\max\limits_{s\in\{s_1,s_2\}} \max\limits_{v\in\mathrm{V}} \min\limits_{p\in\mathsf{Paths}(s,v)} \mathsf{length}(p)}{\min\limits_{s\in\{s_1,s_2\}} \max\limits_{v\in\mathrm{V}} \min\limits_{p\in\mathsf{Paths}(s,v)} \mathsf{length}(p)} && \text{Similar to Fig. 2 for } \textsc{Radius} \text{ in the paper.} \\
&= \frac{\left(\begin{array}{l} \mathsf{ilet}\ \langle x_1, y_1\rangle \coloneqq \min\limits_{\langle s_1,s_2\rangle} \mathcal{F}\ \mathsf{in} \\ \mathsf{mlet}\ \langle x_1', y_1'\rangle \coloneqq \langle x_1, y_1\rangle\ \mathsf{in} \\ \mathsf{rlet}\ \langle x_1'', y_1''\rangle \coloneqq \mathcal{R}\ \langle x_1', y_1'\rangle\ \mathsf{in} \\ \max(x_1'', y_1'') \end{array}\right)}{\left(\begin{array}{l} \mathsf{ilet}\ \langle x_2, y_2\rangle \coloneqq \min\limits_{\langle s_1,s_2\rangle} \mathcal{F}\ \mathsf{in} \\ \mathsf{mlet}\ \langle x_2', y_2'\rangle \coloneqq \langle x_2, y_2\rangle\ \mathsf{in} \\ \mathsf{rlet}\ \langle x_2'', y_2''\rangle \coloneqq \mathcal{R}\ \langle x_2', y_2'\rangle\ \mathsf{in} \\ \min(x_2'', y_2'') \end{array}\right)} \ \text{where} \ \begin{array}{l} \mathcal{F} \coloneqq \lambda p.\ \langle \mathsf{length}(p), \mathsf{length}(p)\rangle \\ \mathcal{R}\ (\langle a, b\rangle, \langle a', b'\rangle) \coloneqq \\ \quad \langle \max(a, a'), \max(b, b')\rangle \end{array} && \text{By F}\textsc{LetsBin} \\
&= \left(\begin{array}{l} \mathsf{ilet}\ \langle\langle x_1, y_1\rangle, \langle x_2, y_2\rangle\rangle \coloneqq \langle \min\limits_{\langle s_1,s_2\rangle} \mathcal{F}, \min\limits_{\langle s_1,s_2\rangle} \mathcal{F}\rangle\ \mathsf{in} \\ \mathsf{mlet}\ \langle\langle x_1', y_1'\rangle, \langle x_2, y_2\rangle\rangle \coloneqq \langle\langle x_1, y_1\rangle, \langle x_2, y_2\rangle\rangle\ \mathsf{in} \\ \mathsf{rlet}\ \langle\langle x_1'', y_1''\rangle, \langle x_2'', y_2''\rangle\rangle \coloneqq \langle \mathcal{R}\ \langle x_1', y_1'\rangle, \mathcal{R}\ \langle x_2', y_2'\rangle\rangle\ \mathsf{in} \\ \max(x_1'', y_1'')/\min(x_2'', y_2'') \end{array}\right) && \begin{array}{r} \text{By IE}\textsc{lim} \\ \text{Common} \\ \text{path-based} \\ \text{reduction} \\ \text{elimination} \end{array} \\
&= \left(\begin{array}{l} \mathsf{ilet}\ \langle x_1, y_1\rangle \coloneqq \min\limits_{\langle s_1,s_2\rangle} \mathcal{F}\ \mathsf{in} \\ \mathsf{mlet}\ \langle\langle x_1', y_1'\rangle, \langle x_2, y_2\rangle\rangle \coloneqq \langle\langle x_1, y_1\rangle, \langle x_1, y_1\rangle\rangle\ \mathsf{in} \\ \mathsf{rlet}\ \langle\langle x_1'', y_1''\rangle, \langle x_2'', y_2''\rangle\rangle \coloneqq \langle \mathcal{R}\ \langle x_1', y_1'\rangle, \mathcal{R}\ \langle x_2', y_2'\rangle\rangle\ \mathsf{in} \\ \max(x_1'', y_1'')/\min(x_2'', y_2'') \end{array}\right) && \begin{array}{r} \text{Similarly,} \\ \text{by Common} \\ \text{vertex-based} \\ \text{mapping} \\ \text{elimination} \end{array} \\
&= \left(\begin{array}{l} \mathsf{ilet}\ \langle x_1, y_1\rangle \coloneqq \min\limits_{\langle s_1,s_2\rangle} \mathcal{F}\ \mathsf{in} \\ \mathsf{mlet}\ \langle x_1', y_1'\rangle \coloneqq \langle x_1, y_1\rangle\ \mathsf{in} \\ \mathsf{rlet}\ \langle\langle x_1'', y_1''\rangle, \langle x_2'', y_2''\rangle\rangle \coloneqq \langle \mathcal{R}\ \langle x_1', y_1'\rangle, \mathcal{R}\ \langle x_1', y_1'\rangle\rangle\ \mathsf{in} \\ \max(x_1'', y_1'')/\min(x_2'', y_2'') \end{array}\right) && \begin{array}{r} \text{Similarly,} \\ \text{by Common} \\ \text{vertex-based} \\ \text{reduction} \\ \text{elimination} \end{array} \\
&= \left(\begin{array}{l} \mathsf{ilet}\ \langle x_1, y_1\rangle \coloneqq \min\limits_{\langle s_1,s_2\rangle} \mathcal{F}\ \mathsf{in} \\ \mathsf{mlet}\ \langle x_1', y_1'\rangle \coloneqq \langle x_1, y_1\rangle\ \mathsf{in} \\ \mathsf{rlet}\ \langle x_1'', y_1''\rangle \coloneqq \mathcal{R}\ \langle x_1', y_1'\rangle\ \mathsf{in} \\ \max(x_1'', y_1'')/\min(x_1'', y_2'') \end{array}\right) \ \text{where} \ \begin{array}{l} \mathcal{F} \coloneqq \lambda p.\ \langle \mathsf{length}(p), \mathsf{length}(p)\rangle \\ \mathcal{R}\ (\langle a, b\rangle, \langle a', b'\rangle) \coloneqq \\ \quad \langle \max(a, a'), \max(b, b')\rangle \end{array}
\end{aligned}
$$

3 Mapping Specification to Iteration-Map-Reduce

3.1 Iterative Reduction and its Correctness

We consider four variants of iterative reduction based on whether the values of the predecessors are pulled by the vertex itself or pushed by the predecessors, and whether the reduction function $\mathcal{R}$ is idempotent.

3.1.1 Pull Model

Pull model with idempotent reduction.

THEOREM 8 (CORRECTNESS OF PULL (IDEMPOTENT REDUCTION)). *For all* $\mathcal{R}$, $\mathcal{F}$, C, $\mathcal{I}$, $\mathcal{P}$, *and* $k \geq 1$, *if the conditions* $\mathbb{C}_1$ - $\mathbb{C}_9$ *hold, then* $\mathcal{S}^k_{\text{pull+}}(v) = \mathcal{S}pec^k(v)$.

The full proof is available in the appendix § 4.4.1. We prove by induction that after each iteration k, the value $\mathcal{S}^k_{\text{pull+}}(v)$ of each vertex v is $\mathcal{S}pec^k(v)$ that is the reduction over paths to v of length less than k. At the iteration $k = 1$, the specification $\mathcal{S}pec^1(v)$ requires reduction on only the paths of length zero to each vertex. Therefore, by the conditions $\mathbb{C}_1$ - $\mathbb{C}_2$, the initialization function $\mathcal{I}$ properly initializes each vertex v to $\mathcal{S}pec^1(v)$. In each iteration $k + 1$, if there is any predecessor of the vertex v whose value is changed in the previous iteration k, then their new values are propagated by $\mathcal{P}$ and reduced together by $\mathcal{R}$ and then reduced with the current value of v. By the conditions $\mathbb{C}_7$ and $\mathbb{C}_8$, the reduction function $\mathcal{R}$ is commutative and associative, and can be applied to the propagated values in any order. By the induction hypothesis, the value of each predecessor u is the reduction of the paths to u of length l, $0 \leq l < k$. The predecessors that have no paths and store $\perp$ are ignored by the conditions $\mathbb{C}_3$ and $\mathbb{C}_6$. By the conditions $\mathbb{C}_4$ and $\mathbb{C}_5$, the propagation of the value of a predecessor u of the vertex v is equal to the reduction over the paths to v that pass through u. Since these paths include at least the edge (from u to v), their length l is $0 < l < k + 1$. The previous value of v itself is the reduction over paths to v of length l, $0 \leq l < k$. Since, the reduction function $\mathcal{R}$ is idempotent, reducing these two values absorbs the values of the repeated paths and results in the reduction over all paths of length l, $0 \leq l < k + 1$. If the value of none of the predecessors is changed in the previous iteration, then the above reduction is skipped, and it can be shown that the current value of the vertex is already equal to the above reduction.

Pull model with non-idempotent reduction.

Theorem 9 (Correctness of Pull (non-idempotent reduction)). *For all* $\mathcal{R}$, $\mathcal{F}$, $\mathcal{I}$, $\mathcal{P}$, $k \geq 1$, *and* s, *let* $C(p) := (\mathsf{head}(p) = s)$, *if the conditions* $\mathbb{C}_1$ - $\mathbb{C}_8$ *hold, and* s *is not on any cycle,* $\mathcal{S}^k_{\mathsf{pull-}}(v) = \mathcal{S}pec^k(v)$.

The full proof is available in the appendix § 4.4.2. The proof of this theorem is similar to the proof of Theorem 8. Based on the induction hypothesis, the reduction of the propagated values covers the paths of length l, $0 < l < k + 1$. The current value of v itself covers the paths of length l, $0 \leq l < k$. Since the two sets of paths overlap and the reduction function may not be idempotent, the reduction with the latter is avoided. However, no path is missed by avoiding the reduction. The difference is only the paths of length 0. The vertices other than the source s do not have a path of length 0 from s. The source s is correctly initialized to the value of $\mathcal{F}$ on the zero-length path $\langle s, s \rangle$ from s to itself, and since s is not on any cycle, its correct value is never overwritten.

3.1.2 Push Model

Push model with idempotent reduction.

THEOREM 10 (CORRECTNESS OF PUSH (IDEMPOTENT REDUCTION)). *For all* $\mathcal{R}$, $\mathcal{F}$, $\mathcal{C}$, $\mathcal{I}$, $\mathcal{P}$, *and* $k \geq 1$, *if the conditions* $\mathbb{C}_1$ - $\mathbb{C}_9$ *hold,* $\mathcal{S}^k_{\text{push+}}(v) = \mathcal{S}pec^k(v)$.

The full proof is available in § 4.4.3. Similar to the proof of Theorem 8, the reduction function should be idempotent since the reduced values may cover overlapping sets of paths. The main difference is that instead of propagating and reducing the values of all the predecessors of v, only the values of the predecessors $\{\overline{u}\}$ of v that have been changed in the previous iteration k are propagated and reduced. Therefore, the values of the unchanged predecessors $\{\overline{w}\}$ of v are not reduced with the current value of v. However, the resulting value of v does not miss any path to v that goes through an unchanged predecessor w. If w is never changed, there is no path from the source(s) to it. If it is changed in the previous iterations, in the last such iteration, its value has been already reduced with the current value of v.

Push model with non-idempotent reduction.

This model works for non-idempotent (in addition to idempotent) reduction functions. We consider two instances of this model: first the basic and then the optimized iteration model.

The first variant of push, non-idempotent was defined in Fig. 8, Def. 4.

THEOREM 11 (CORRECTNESS OF PUSH (NON-IDEMPOTENT REDUCTION) I).
For all $\mathcal{R}$, $\mathcal{F}$, $\mathcal{I}$, $\mathcal{P}$, $k \geq 1$, *and* s, *let* $C(p) \coloneqq (\text{head}(p) = s)$, *if the conditions* $\mathbb{C}_1$ - $\mathbb{C}_8$ *hold, and* s *is not on any cycle,* $\mathcal{S}^k_{\text{push}-}(v) = \mathcal{S}pec^k(v)$

The full proof is available in the appendix § 4.4.4. The proof of this theorem is similar to the proof of Theorem 9. Based on the induction hypothesis, the reduction of the propagated values covers the paths of length l, $0 < l < k + 1$. Let us consider the paths of length 0. The vertices other than the source s do not have a path of length 0 from s. The source s is correctly initialized to the value of $\mathcal{F}$ on the zero-length path $\langle s, s \rangle$ from s to itself, and since s is not on any cycle, its correct value is never overwritten.

The second variant is represented in Def. 7 below. Let the value of the vertex v in the iteration k be represented as $\mathcal{S}^k_{\text{push}-}(v)$. The main difference with the previous model is that every changed predecessor u_i first rollbacks its previous update before applying its new update. The rollback function $\mathcal{B}$, given a value n and an edge $\langle u, v\rangle$ where n is the previous value of u, defines the value that is propagated to v to be rolled back. The rollback value is expected to cancel the previously propagated value. For example, for the PAGERANK use-case as Fig. 7 shows, the rollback function returns the negation of the previously propagated value. For each predecessor u_i, the rollback function $\mathcal{B}$ is applied to the previous value $\mathcal{S}^{k-1}_{\text{push}-}(u_i)$ of u_i and the edge $\langle u_i, v\rangle$, and the propagate function $\mathcal{P}$ is applied to the latest value $\mathcal{S}^k_{\text{push}-}(u_i)$ of u_i and the edge $\langle u_i, v\rangle$. The two resulting values are reduced with the current value of v.

DEFINITION 7 (PUSH (NON-IDEMPOTENT REDUCTION) II).

$$\begin{aligned}
&\mathcal{S}^0_{\text{push}-}(v) := \bot \\
&\mathcal{S}^1_{\text{push}-}(v) := \mathcal{I}(v) \\
&\mathcal{S}^{k+1}_{\text{push}-}(v) := \mathcal{E}(S_n), \quad k \geq 1 \quad \textit{where} \\
&\quad \text{let } \{u_0, .., u_{n-1}\} := \text{CPreds}^k(v) \text{ in} \\
&\quad S_0 := \mathcal{S}^k_{\text{push}-}(v) \\
&\quad S_{i+1} := \mathcal{R}(\mathcal{R}(S_i, \\
&\qquad\quad \mathcal{B}\left(\mathcal{S}^{k-1}_{\text{push}-}(u_i), \langle u_i, v\rangle\right)), \\
&\qquad\quad \mathcal{P}\left(\mathcal{S}^{k}_{\text{push}-}(u_i), \langle u_i, v\rangle\right))
\end{aligned}$$

The correctness of this variant of iteration is dependent on the following condition for the propagation and rollback functions.

$$\begin{aligned}
&\mathbb{C}_{11} \text{ (Rollback)}: \\
&\quad \forall n, n'.\ \mathcal{R}(n, \mathcal{R}(\mathcal{P}(n', e), \\
&\qquad\qquad\qquad \mathcal{B}(n', e))) = n
\end{aligned}$$

As we saw in Def. 7, in this variant of push model with non-idempotent reduction $\mathcal{S}^k_{\text{push}-}(v)$, each predecessor first rollbacks its previously propagated value before propagating its new value. The rollback value is expected to cancel the previously propagated value. This requirement is captured as the condition $\mathbb{C}_{11}$ above. As an example, the number of shortest paths use-case NSP, after fusion, calculates a pair for each vertex where the first element is the shortest path weight and the second element is the number of such paths. For NSP, the propagate function is $\mathcal{P} = \lambda\langle w, n\rangle, e.\ \langle w + \text{weight}(e), n\rangle$ and the rollback function is $\mathcal{B} = \lambda\langle w, n\rangle, e.\ \langle w, -n\rangle$.

For synthesis in this model, after the propagation function $\mathcal{P}$ is synthesized, the condition $\mathbb{C}_{11}$ is used to synthesize the rollback function $\mathcal{B}$.

The following theorem states that if the conditions $\mathbb{C}_1$ - $\mathbb{C}_8$ and the condition $\mathbb{C}_{11}$ hold, this model complies with the specification $\mathcal{S}pec^k(v)$.

THEOREM 12 (CORRECTNESS OF PUSH (NON-IDEMPOTENT REDUCTION) II). *For all $\mathcal{R}, \mathcal{F}, C, \mathcal{I}, \mathcal{P}$, and $k \geq 1$, if the conditions $\mathbb{C}_1$ - $\mathbb{C}_8$ and $\mathbb{C}_{11}$ hold, $\mathcal{S}^k_{\text{push}-}(v) = \mathcal{S}pec^k(v)$.*

The full proof is available in § 4.4.4. First, we show that after each iteration $k+1$, the value of each vertex v is the reduction of its initial value and the value of predecessors in the previous iteration k. Even though only the changed predecessors push values, similar to the proof of Theorem 10, the value of no predecessor is missed. If a predecessor is never changed, it has the value $\bot$ that is ignored in the reduction anyway. If it is changed in the previous iterations, in the last such iteration, its value has been pushed and reduced with the current value of v. Since reduction is not idempotent, each predecessor first rollbacks its old value before applying its new value. Second, using the first fact, we show by induction that the value of each vertex v is the reduction of the paths to v of length less than $k+1$. Similar to the previous proofs, it can be shown that the initial value of v is the result of reduction on paths to v of length 0. Further, using the induction hypothesis, it can be shown that the propagation of values from the predecessors in iteration $k+1$ results in the reduction over paths to v of length l, $0 < l < k+1$. Reducing the two values results in the reduction over paths to v of length l, $0 \leq l < k+1$ that the specification $\mathcal{S}pec^{k+1}(v)$ requires.

Number of Shortest Paths (NSP)

$$
\begin{array}{lcl}
\mathcal{I} & := & \lambda v.\ \ \text{if } (v = s)\ \langle 0, 1\rangle \text{ else } \bot \\
\mathcal{P} & := & \lambda n, e.\ \ n + \text{weight}(e) \\
\mathcal{R} & := & \lambda \langle w, n\rangle, \langle w', n'\rangle. \\
 & & \quad \text{if } (w = w')\ \langle w, n + n'\rangle \\
 & & \quad \text{elseif } (w > w')\ \langle w', n'\rangle \\
 & & \quad \text{else } \langle w, n\rangle \\
\mathcal{E} & := & \lambda n.\ \ n \\
\mathcal{B} & := & \lambda \langle w, n\rangle, e.\ \langle w, -n\rangle
\end{array}
$$

Fig. 16. The number of shortest paths

3.1.3 Asynchronous Model

$$\begin{aligned}&\text{The predecessors of the vertex } v \text{ that changed value in the iteration } k\text{:}\\ &\mathsf{CPreds}^k(v) = \left\{u \mid u \in \mathsf{preds}(v) \wedge \mathcal{S}^k(u) \neq \mathcal{S}^{k-1}(u)\right\}\end{aligned}$$

Definition 8 (Pull (idempotent reduction)).

$$\begin{aligned}
\mathcal{S}^0_{\mathsf{apull+}}(v) &:= \bot\\
\mathcal{S}^1_{\mathsf{apull+}}(v) &:= \mathcal{I}(v)\\
\mathcal{S}^{k+1}_{\mathsf{apull+}}(v) &:= \begin{cases}\mathcal{S}^k_{\mathsf{apull+}}(v) & \textit{if } \mathsf{CPreds}^k(v) = \emptyset\\ \mathcal{E}\left[\mathcal{R}\left(\mathcal{S}^k_{\mathsf{apull+}}(v),\ \mathcal{R}_{u\in\mathsf{preds}(v)}\,\mathcal{P}\left(\mathcal{S}^k_{\mathsf{apull+}}(u)\ ?\ \mathcal{S}^{k+1}_{\mathsf{apull+}}(u), \langle u, v\rangle\right)\right)\right] & \textit{else}\end{cases} \qquad k \geq 1
\end{aligned}$$

Definition 9 (Pull (non-idempotent reduction)).

$$\begin{aligned}
\mathcal{S}^0_{\mathsf{apull-}}(v) &:= \bot\\
\mathcal{S}^1_{\mathsf{apull-}}(v) &:= \mathcal{I}(v)\\
\mathcal{S}^{k+1}_{\mathsf{apull-}}(v) &:= \begin{cases}\mathcal{S}^k_{\mathsf{apull-}}(v) & \textit{if } \mathsf{CPreds}^k(v) = \emptyset\\ \mathcal{E}\left[\mathcal{R}_{u\in\mathsf{preds}(v)}\,\mathcal{P}\left(\mathcal{S}^k_{\mathsf{apull-}}(u)\ ?\ \mathcal{S}^{k+1}_{\mathsf{apull-}}(u), \langle u, v\rangle\right)\right] & \textit{else}\end{cases} \qquad k \geq 1
\end{aligned}$$

Definition 10 (Push (idempotent reduction)).

$$\begin{aligned}
&\mathcal{S}^0_{\mathsf{apush+}}(v) := \bot\\
&\mathcal{S}^1_{\mathsf{apush+}}(v) := \mathcal{I}(v)\\
&\mathcal{S}^{k+1}_{\mathsf{apush+}}(v) := \mathcal{E}(S_n), \quad k \geq 1 \quad \textit{where}\\
&\quad \mathsf{let}\ \{u_0, .., u_{n-1}\} := \mathsf{CPreds}^k(v)\ \mathsf{in}\\
&\quad \mathsf{let}\ m_i := |\mathsf{CPreds}^k(u_i)|\ \mathsf{in}\\
&\quad S_0(v) := \mathcal{S}^k_{\mathsf{apush+}}(v)\\
&\quad S_{i+1}(v) := \mathcal{R}\left(S_i(v),\ \mathcal{P}\left(?_{j\in\{1..m_i\}}\, S_j(u_i), \langle u_i, v\rangle\right)\right)
\end{aligned}$$

Definition 11 (Push (non-idempotent reduction)).

$$\begin{aligned}
&\mathcal{S}^0_{\mathsf{apush-}}(v) := \bot\\
&\mathcal{S}^1_{\mathsf{apush-}}(v) := \mathcal{I}(v)\\
&\mathcal{S}^{k+1}_{\mathsf{apush-}}(v) := \mathcal{E}(S_n(v)), \quad k \geq 1 \quad \textit{where}\\
&\quad \mathsf{let}\ \{u_0, .., u_{n-1}\} := \mathsf{CPreds}^k(v)\ \mathsf{in}\\
&\quad S_0(v) := \mathcal{S}^k_{\mathsf{apush-}}(v)\\
&\quad S_{i+1}(v) := \mathcal{R}(\mathcal{R}(S_i(v),\\
&\qquad\quad \mathcal{B}\left(b^{k-1}(u_i), \langle u_i, v\rangle\right)),\\
&\qquad\quad \mathcal{P}\left(b^k(u_i), \langle u_i, v\rangle\right))\\
&\quad b^{k+1}(v) := ?_{i\in\{0..n\}}\, S_i(v)
\end{aligned}$$

Fig. 17. Four Iterative Reduction Methods (in the asynchronous mode). The operator ? is the non-deterministic choice operator.

The iteration models that were presented in Fig. 8 are *synchronous*. In the synchronous model, in each iteration $k+1$, each vertex v stores both its previous value $\mathcal{S}^k(v)$ and its new value $\mathcal{S}^{k+1}(v)$. The previous value $\mathcal{S}^k(v)$ of v is propagated to update other vertices and updates to the value of v are stored in its new value $\mathcal{S}^{k+1}(v)$. Therefore, the updates in the current iteration do not affect the values that are propagated. In the *asynchronous* model, however, each vertex stores one value. The single value is used to both propagate the current value of the vertex and store its new value.

Asynchronous model can save space and converge faster but is more subtle. The values that are propagated in iteration $k + 1$ can be either the previous value $\mathcal{S}^k(v)$, the old value $\mathcal{S}^{k+1}(v)$ or an intermediate value between the two. The high-level idea is that the new value has more information than the old value i.e. covers more paths. Thus, vertices reach convergence faster.

The asynchronous pull model for idempotent and non-idempotent reduction functions are presented in Def. 8 and Def. 9. They are very similar to the corresponding synchronous pull models that were presented in Def. 1 and Def. 2. Now, the propagated value is either the previous value $\mathcal{S}^k_{\mathsf{apull+}}(u)$ or the new value $\mathcal{S}^{k+1}_{\mathsf{apull+}}(u)$ of the predecessor u. The operator ? is the non-deterministic choice operator that non-deterministically returns one if its operands.

The asynchronous push model for idempotent reduction functions is presented in Def. 10. It is similar to the corresponding synchronous definition presented in Def. 3. The difference is that instead of the previous value $\mathcal{S}^k_{\mathsf{push+}}(u_i)$ of each predecessor u_i, one of its intermediate values $S_j(u_i)$ is propagated. Assuming that the predecessor u_i has m_i changed predecessors itself, u_i has the intermediate values $S_j(u)$ where $j \in \{1..m_i\}$, one after each push from its predecessors. The value propagated to v can non-deterministically be any of the intermediate values.

The asynchronous push model for non-idempotent reduction functions is presented in Def. 11. It not similar to the corresponding synchronous definition presented in Def. 4. The difference is that the values propagated by a vertex can be any of its intermediate values and not necessarily its value at the end of the last iteration. Thus, we need to store the previously propagated values to roll them back before propagating new values. Consider a vertex v and its predecessor u_i. The value that u_i propagates to v in iteration k is stored as $b^k(u_i)$. In iteration $k + 1$, to push from the predecessor u_i to the vertex v, the value $b^{k-1}(u_i)$ is rolled back by the rollback function $\mathcal{B}$ and the new value $b^k(u_i)$ is propagated by the propagation function $\mathcal{P}$.

We define $P^\infty(v)$ as all the paths to the vertex v (that satisfy the condition C).

Definition 12 (Paths). $P^\infty(v) = \{p \mid p \in \mathsf{Paths}(v) \ \wedge \ C(p)\}$

The definition of specification $\mathcal{S}pec(v)$ is the same as definition Def. 5; only the paths are factored to $P^\infty(v)$.

Definition 13 (Specification). $\mathcal{S}pec(v) = \mathcal{R}_{p \in P^\infty(v)} \ \mathcal{F}(p)$

We define $P^k(v)$ as the paths to the vertex v of length less than k (that satisfy the condition C).

Definition 14 (k-Paths). $P^k(v) = \{p \mid p \in \mathsf{Paths}(v) \ \wedge \ C(p) \ \wedge \ \mathsf{length}(p) < k\}$

The definition of specification for iteration k, $\mathcal{S}pec^k(v)$, is the same as definition Def. 6; only the paths are factored to $P^k(v)$.

Definition 15 (k-Specification). $\mathcal{S}pec^k(v) = \mathcal{R}_{p \in P^k(v)} \ \mathcal{F}(p)$

Since in the asynchronous model, in an iteration k, the value of vertices may cover paths of length k or longer, we define $aP^k(v)$ as the set of paths that include paths of length less than k and maybe more.

Definition 16 (a-k-Paths). $aP^k(v) = \{P \mid P(k) \subseteq P \subseteq P^\infty(v)\}$

Since in the asynchronous model, vertices may propagate any one of the multiple intermediate values, we define asynchronous specification for iteration k, $a\mathcal{S}pec^k(v)$, as set of values: the reductions of any set of paths P in $aP^k(v)$.

Definition 17 (a-k-Specification). $a\mathcal{S}pec^k(v) = \{\mathcal{R}_{p \in P} \ \mathcal{F}(p) \mid P \in aP^k(v)\}$

All the asynchronous models presented in Fig. 17 comply with the asynchronous specification. In each iteration, the value stored at vertex v is in the set of values $a\mathcal{S}pec^k(v)$.

Theorem 13 (Correctness of Pull (idempotent reduction)). *Forall* $\mathcal{R}, \mathcal{F}, C, \mathcal{I}, \mathcal{P}$, *and* $k \geq 1$, *if the conditions* $\mathbb{C}_1$ - $\mathbb{C}_4$ *and* $\mathbb{C}_6$ - $\mathbb{C}_9$ *hold, then* $\mathcal{S}^k_{\text{apull+}}(v) \in a\mathcal{S}pec^k(v)$

The proof is similar to the proof of Theorem 8. The set of paths covered by $\mathcal{S}^{k+1}_{\text{apull+}}(u)$ is a superset of path covered by $\mathcal{S}^k_{\text{apull+}}(u)$. The reduction over the set of paths in the difference is factored out in the proof.

Theorem 14 (Correctness of Pull (non-idempotent reduction)). *Forall* $\mathcal{R}, \mathcal{F}, \mathcal{I}, \mathcal{P}, k \geq 1$, *and* s, *let* $C(p) := (\text{head}(p) = s)$, *if the conditions* $\mathbb{C}_1$ - $\mathbb{C}_4$ *and* $\mathbb{C}_6$ - $\mathbb{C}_8$ *hold, and* s *is not on any cycle,* $\mathcal{S}^k_{\text{apull}-}(v) \in a\mathcal{S}pec^k(v)$

The proof is similar to the proof of Theorem 9. The set of paths covered by $\mathcal{S}^{k+1}_{\text{apull+}}(u)$ is a superset of path covered by $\mathcal{S}^k_{\text{apull+}}(u)$. The reduction over the set of paths in the difference is factored out in the proof.

Theorem 15 (Correctness of Push (idempotent reduction)). *Forall* $\mathcal{R}, \mathcal{F}, C, \mathcal{I}, \mathcal{P}$, *and* $k \geq 1$, *if the conditions* $\mathbb{C}_1$ - $\mathbb{C}_4$ *and* $\mathbb{C}_6$ - $\mathbb{C}_9$ *hold,* $\mathcal{S}^k_{\text{apush+}}(v) \in a\mathcal{S}pec^k(v)$

The proof is similar to the proof of Theorem 10. The set of paths covered by $S_j(u_i)$ is a superset of path covered by $\mathcal{S}^k_{\text{push+}}(u_i)$. The reduction over the set of paths in the difference is factored out in the proof.

Theorem 16 (Correctness of Push (non-idempotent reduction)). *Forall* $\mathcal{R}, \mathcal{F}, C, \mathcal{I}, \mathcal{P}$, *and* $k \geq 1$, *if the conditions* $\mathbb{C}_1$ - $\mathbb{C}_8$ *hold,* $\mathcal{S}^k_{\text{apush}-}(v) \in a\mathcal{S}pec^k(v)$

The proof is similar to the proof of Theorem 25. The set of paths covered by $b_k(u_i)$ is a superset of path covered by $\mathcal{S}^k_{\text{pull}-}(u_i)$. The reduction over the set of paths in the difference is factored out in the proof.

Theorem 17 (Termination). *Forall* $\mathcal{R}, \mathcal{F}$, *and* C, *if the graph is acyclic or the condition* $\mathbb{C}_{10}$ *holds, then there exists* k' *such that for every* $k \geq k'$, $a\mathcal{S}pec^k(v) = \{\mathcal{S}pec(v)\}$.

The proof is similar to the proof of Theorem 27. Let l be the longest simple path to v. If the graph is acyclic, there is no path longer than l. Thus, for any $k > l + 1$, $P^k(v) = \{P^\infty(v)\}$. Therefore, $a\mathcal{S}pec^k(v) = \{\mathcal{S}pec(v)\}$. Even if the graph is cyclic, for any path p longer than l, the condition $\mathbb{C}_{10}$ states that reducing the value of p with the value of simple(p) leaves the value of simple(p) unchanged. Thus, $\mathcal{R}_{p \in P^k(v)} \mathcal{F}(p) = \mathcal{R}_{p \in P^{l+1}(v)} \mathcal{F}(p)$. Thus, $a\mathcal{S}pec^k(v) = \{\mathcal{R}_{p \in P^{l+1}(v)} \mathcal{F}(p)\}$. Similarly, it can be shown that $\mathcal{S}pec(v) = \{\mathcal{R}_{p \in P^{l+1}(v)} \mathcal{F}(p)\}$. Therefore, $a\mathcal{S}pec^k(v) = \{\mathcal{S}pec(v)\}$.

An immediate corollary of the above theorem is that if the graph is acyclic or the condition $\mathbb{C}_{10}$ holds, then all the four asynchronous iteration models eventually terminate and converge to the specification (if their corresponding conditions in Theorem 13 to Theorem 16 hold). For example the corollary for the asynchronous pull model for idempotent reduction functions is the following. The corollary for the other models is similar.

Corollary 18 (Termination). *Forall* $\mathcal{R}, \mathcal{F}, C, \mathcal{I}$, *and* $\mathcal{P}$, *if the conditions* $\mathbb{C}_1$ - $\mathbb{C}_4$ *and* $\mathbb{C}_6$ - $\mathbb{C}_9$ *hold and the graph is acyclic or the condition* $\mathbb{C}_{10}$ *holds, then there exists an iteration* k *such that* $\mathcal{S}^k_{\text{apull+}}(v) = \mathcal{S}pec(v)$

3.1.4 *Streaming Graphs*

In contrast to a static graph, a streaming graph can continuously change in response to external events. Thus, to have up-to-date results, the graph analytics computations should be periodically repeated. Stream graph processing strives to benefit from the results computed prior to the updates instead of restarting the iteration form the initial values. The idea is that starting from the prior result can accelerate the convergence. What are the conditions such that the incremental computation yields the correct results? We first consider addition and then removal of edges and present the correctness conditions for incremental commutation after each.

Incremental Computation. Consider a graph G. Let us denote the result of a path-based reduction $\mathcal{S}pec(v)$ on G as $\mathcal{S}pec_G(v)$. Let $G + \delta$ be the result of updating (adding or removing) an edge $e = \langle s_e, t_e \rangle$ in G. The incremental computation on $G + \delta$ starts from the prior result $\mathcal{S}pec_G(v)$ for G. The incremental pull model is similar to the basic model (of Def. 1). The difference is that (1) the starting state is $\mathcal{S}pec_G(v)$ instead of $\mathcal{I}(v)$ except for the sink node t_e and if the update is a removal, and (2) that the vertex t_e is updated in the starting iteration. Thus, the state of the incremental computation at iteration k denoted as $\mathcal{S}^k_{G+\delta}(v)$ is defined as follows:

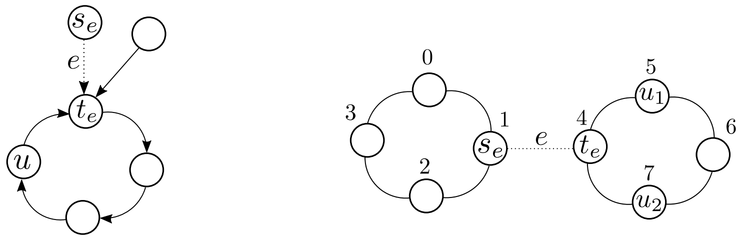
Definition 18 (Incremental pull model (with idempotent reduction)).

$$
\begin{aligned}
\mathcal{S}^1_{G+\delta}(v) \quad &:= \quad \begin{cases} \mathcal{I}(v) & \textit{if } (\delta \textit{ is removal}) \wedge (v = t_e) \\ \mathcal{S}pec_G(v) & \textit{else} \end{cases} \\
\mathcal{S}^{k+1}_{G+\delta}(v) \quad &:= \quad \begin{cases} \mathcal{R}\left[\mathcal{S}^k_{G+\delta}(v),\ \mathcal{R}_{u \in \mathsf{preds}(v)}\ \mathcal{P}\Big(\mathcal{S}^k_{G+\delta}(u), \langle u, v\rangle\Big)\right] & \textit{if } (k = 1 \wedge v = t_e) \vee (\mathsf{CPreds}^k(v) \neq \emptyset) \\ \mathcal{S}^k_{G+\delta}(v) & \textit{else} \end{cases} \quad k \geq 1
\end{aligned}
$$

Addition of Edges. If the update δ in $G + \delta$ is adding an edge, does the result of incremental computation $\mathcal{S}^k_{G+\delta}(v)$ converge to its specification $\mathcal{S}pec(v)$? It turns out that it does with the same conditions as the static case. Adding an edge only increases the set of paths. The prior value of a vertex is the result of reduction on the old set of paths to that vertex. That set may now be incomplete. However, the prior values can help the incremental computation skip most of the initial iterations. For example, in the shortest path SSSP use-case, the newly added edge may improve the previously found shortest path only for some of the vertices. Subsequent iterations will eventually reduce the values of all the new paths with the prior values of the vertices. As the reduction function is assumed to be commutative, associative and idempotent, the reduction order and repeated reductions of a path do not affect the result. Thus, we can state the following theorem for the correctness of incremental reduction after adding edges.

Theorem 19 (Correctness after adding edges). *For all $\mathcal{R}$, $\mathcal{F}$, $\mathcal{I}$ and $\mathcal{P}$, if the conditions $\mathbb{C}_1$ - $\mathbb{C}_{10}$ hold and the update δ is addition of an edge, then there exists k such that $\mathcal{S}^k_{G+\delta}(v) = \mathcal{S}pec(v)$.*

Removal of Edges. In contrast to adding, if the update is removing an edge, the incremental computation is not necessarily correct. When an edge $\langle s_e, t_e \rangle$ is removed, the value of t_e becomes incorrect if it has been calculated using the value of s_e. Thus, the incremental computation (Def. 18) recalculates the value of t_e based on the values of its remaining predecessors. The intention is that this update calculates the correct value of t_e. However, as Fig. 18a shows, if there is a loop from t_e back to one of its predecessors u, and the value of u has been calculated based on the old value of t_e, the recalculated value of t_e is still incorrect. The new value of t_e can lead to calculation of new values back to u and then again for t_e. The question is whether the iterative calculation around the loop eventually forgets the incorrect value. It turns out that it does, if extending a path with an edge makes the value of the path less favorable during reduction. For example, in the SSSP use-case, the value of a path is its weight and the weight of an extended path increases; thus, the extended path is less favorable for the min reduction function. The cycle can take only larger values back to



(a) Removing an edge and the subsequent update through a cycle

(b) Removing an edge in the CC use-case

Fig. 18. Removing edges

t_e through the predecessor u, and eventually, the values coming from the other predecessors will be smaller and thus, chosen by the min reduction function. Thus, the incremental computation for the shortest path use-case SSSP will eventually converge to the correct values.

However, in the CC use-case, the value of a path is the identifier of its source; thus, the value of an extended path stays the same. Consider the graph in Fig. 18b where two cycles are connected by the edge $e = \langle s_e, t_e \rangle$ where the cycle on the s_e side has the vertex with the smallest identifier 0. The iteration for G results in 0 as the component identifier of all vertices. Upon the removal of e, the neighbors u of t_e in the loop continue feeding 0 back to t_e which prevents spreading the larger identifier 4 in the cycle. Vertices adopt smaller identifier from their neighbors. The iteration incorrectly converges to 0 as the component identifier of the cycle. We have captured the above sufficient condition in Fig. 19 as the worsening property $\mathbb{C}_{12}$. Extending a path p with an edge e should result in an unequal and worse value. Thus, we can state the following theorem for the correctness of incremental reduction after removing edges.

Streaming:

$\mathbb{C}_{12}$ (Worsening) :

$$\forall p, e.\ \mathcal{R}(\mathcal{F}(p), \mathcal{F}(p \cdot e)) = \mathcal{F}(p) \neq \mathcal{F}(p \cdot e)$$

Fig. 19. Correctness and Termination Conditions

THEOREM 20 (CORRECTNESS AFTER REMOVING EDGES). *For all* $\mathcal{R}$, $\mathcal{F}$, $\mathcal{I}$ *and* $\mathcal{P}$, *if the conditions* $\mathbb{C}_1$ - $\mathbb{C}_{10}$ *and* $\mathbb{C}_{12}$ *hold and the update* δ *is removal of an edge, then there exists* k *such that* $\mathcal{S}^k_{G+\delta}(v) = \mathcal{S}pec(v)$.

However, if the condition $\mathbb{C}_{12}$ does not hold, then all the prior values cannot be simply used and the value of all vertices that are reachable form the vertex t_e should be reset to their initial values. In the example of the CC use-case above, the values of the vertices in the cycle are all reset to their own identifiers. The iteration then correctly converges to the smallest identifier in the cycle. As an optimization, the dependencies between the value of vertices can be tracked at runtime and the values of only the vertices that are dependent on t_e should be reset.

3.1.5 Factored Path-based Reductions

Consider the factored path-based reduction $\underset{c}{\mathcal{R}}\,\mathcal{F}$ with a general configuration c. We show that the correctness conditions for its iterative execution are captured by the conditions that were presented in Fig. 13.

Let us define C^c as follows:

$$\begin{array}{rcl} C^s(p) & := & \mathsf{head}(p) = s \\ C^\perp(p) & := & \mathsf{True} \end{array} \tag{1}$$

Let us define $\mathcal{F}^c$ as follows:

$$\begin{array}{rcl} \langle \mathcal{F}_1, \mathcal{F}_2 \rangle^{\langle c_1, c_2 \rangle}(p) & := & \langle \mathcal{F}_1^{c_1}(p), \mathcal{F}_2^{c_2}(p) \rangle \\ \mathcal{F}^c(p) & := & \mathsf{if}\ (C^c(p))\ \mathcal{F}(p)\ \mathsf{else}\ \perp \end{array} \tag{2}$$

The specification of the factored path-based reduction $\underset{c}{\mathcal{R}}\,\mathcal{F}$ is the following:

$$\mathcal{R}_{p\ \in \mathsf{Paths}(v)}\ \mathcal{F}^c(p)$$

that can be captured by the specification $\mathcal{S}pec(v)$ defined in Def. 5 with $C(p)$ instantiated with True and $\mathcal{F}$ instantiated with $\mathcal{F}^c$.

Thus, the correctness conditions for the factored path-based reduction $\underset{c}{\mathcal{R}}\,\mathcal{F}$ can be captured by the conditions that were presented in Fig. 13 with $C(p)$ instantiated with True and $\mathcal{F}$ instantiated with $\mathcal{F}^c$. In particular, the initialization condition $\mathbb{C}_2$ is trivial and $\mathbb{C}_1$ is simplified to

$$\mathcal{I}(v) = \mathcal{F}^c(\langle v, v \rangle) \tag{3}$$

For example, by Eq. 3 and Eq. 2, for a path-based reduction $\underset{\langle c_1, c_2 \rangle}{\mathcal{R}}\langle \mathcal{F}_1, \mathcal{F}_2 \rangle$, the initialization conditions are the following

$$\begin{array}{l} \forall v.\ C^{c_1}(\langle v, v \rangle) \longrightarrow \mathsf{fst}(\mathcal{I}(v)) = \mathcal{F}_1(\langle v, v \rangle) \\ \forall v.\ \neg C^{c_1}(\langle v, v \rangle) \longrightarrow \mathsf{fst}(\mathcal{I}(v)) = \perp \\ \forall v.\ C^{c_2}(\langle v, v \rangle) \longrightarrow \mathsf{snd}(\mathcal{I}(v)) = \mathcal{F}_2(\langle v, v \rangle) \\ \forall v.\ \neg C^{c_2}(\langle v, v \rangle) \longrightarrow \mathsf{snd}(\mathcal{I}(v)) = \perp \end{array} \tag{4}$$

This means that the initialization for each element of the state tuple mirrors the initialization conditions $\mathbb{C}_1$ and $\mathbb{C}_2$.

3.2 Synthesis of Iterative Reduction

To find candidate expressions for the body of the kernel functions, we apply a type-guided enumerative search to the expression grammar presented in Fig. 20b. The expression constructors have union types; for example, the plus operator + can be applied to both integers Int and floating point Float numbers. The procedure Candidates in Fig. 20a returns the set of expressions of the input type T and size $size$. It is a recursive procedure that uses memoization to avoid redundant enumeration. It keeps a map from types to maps from sizes to the set of previously synthesized expressions. To synthesize an expression of type T, it only considers the expression constructors with the return type T. A constructor itself uses one unit of size. For each constructor c, the Candidates procedure considers all the possible distributions of the remained size, that is $size - 1$, between the parameters of c. For each distribution, it recursively obtains a set of expressions E_i for each parameter p_i using its type and its allocated size. It then applies c to each element of the product of the sets E_i to yield candidate expressions. It memoizes and returns the set of these candidates.

The functions Fig. 20c and Fig. 20d synthesize the functions $\mathcal{I}$ and $\mathcal{P}$. We consider synthesis for $\mathcal{P}$; synthesis for $\mathcal{I}$ is similar. Fig. 20d presents the Synth$\mathcal{P}$ procedure that given the path function $\mathcal{F}$ and the reduction function $\mathcal{R}$ of a path-based reduction, synthesizes the propagation function $\mathcal{P}$. It starts by memoizing expressions of size one, variables and literals, to make them available for the synthesis of the body of $\mathcal{P}$. Let T be the return type of $\mathcal{F}$; vertices store values of type T. The propagation function $\mathcal{P}$ takes a value stored at a vertex (of type of T) and an edge (of type Edge) and returns a vertex value (of type T). Thus, the two input variables of the two input types, the variable n of type T and the variable l of type Edge, are memoized as available expressions. Then, candidate expressions of type T are obtained from the Candidates procedure. Expressions of larger sizes are incrementally checked as candidate bodies for $\mathcal{P}$.

A candidate propagation function $\lambda n, l.\ e$ is correct if the conditions $\mathbb{C}_4$ and $\mathbb{C}_5$ are valid when $\mathcal{P}$ is replaced by the candidate. We use the notation of $\mathcal{A} \vdash \mathcal{A}'$ to represent whether the assertion $\mathcal{A}'$ is valid in the context of the assumed assertion(s) $\mathcal{A}$. To check the validity of an assertion, we use off-the-shelf SMT solvers to check the satisfiability of its negation. The context of the validity check $\mathcal{F}; \mathcal{R}; \Gamma$ is the definition of the functions $\mathcal{F}$ and $\mathcal{R}$ from the given path-based reduction, and a set of assertions Γ that define basic graph functions and relations.

Fig. 21 represents the context assertions Γ: assertions for the path functions length, weight, punultimate and capacity. We define graph functions and relations in the combination of the quantified uninterpreted functions and list theories. We represent a path P as a list of vertices V. The edge weight eweight is a function on pairs of vertices $\langle \mathsf{V}, \mathsf{V} \rangle$ and the path weight weight is a function on paths P to natural numbers $\mathbb{N}$. If the list for the path is empty or has a single vertex, the weight of the path is trivially zero; otherwise, the weight of the path is recursively the sum of the weight of the path without the last edge and the edge weight of the last edge.

For the push model with non-idempotent reduction (Def. 7), after the propagation function $\mathcal{P}$ is synthesized, the condition $\mathbb{C}_{11}$ is used to synthesize the rollback function $\mathcal{B}$.

We note that since the path functions $\mathcal{F}$ never return none $\perp$, the reduction function $\mathcal{R}'$ is simplified to $\mathcal{R}$ in the condition $\mathbb{C}_4$.

```
def Candidates($T$, $size$)
  if (already memoized $E$
      for $T$ and $size$)
    return $E$
  $E \leftarrow \emptyset$
  foreach (expression constructor $c$
    with the return type $T$)
    foreach (distribution $\overline{s_i}$ of $size - 1$
      between parameters $\overline{p_i}$ of $c$)
      foreach ($p_i$ with type $T_i$)
        $E_i \leftarrow$ Candidates($T_i, s_i$)
      $E \leftarrow E \cup \{c(\overline{e}) \mid \overline{e} \in \times \overline{E_i}\}$
  memoize $E$ for $T$ and $size$
  return $E$
```

(a) Type-guided expression enumeration

$$\begin{array}{llll}
e & ::= & n \mid v & \text{Exp} \\
 & \mid & e + e \mid e - e & \\
 & \mid & e = e \mid e < e & \\
 & \mid & \mathsf{min}(e, e) \mid \mathsf{max}(e, e) & \\
 & \mid & \mathsf{if}\ (e)\ \mathsf{then}\ e\ \mathsf{else}\ e & \\
 & \mid & \mathsf{weight}(e) \mid \mathsf{capacity}(e) & \\
 & \mid & \mathsf{indeg}(e) \mid \mathsf{outdeg}(e) & \\
 & \mid & \mathsf{src}(e) \mid \mathsf{dst}(e) & \\
 & \mid & |V| & \\
n & ::= & 0 \mid 1 \mid .. \mid \mathsf{True} \mid \mathsf{False} & \text{Literal} \\
v & & & \text{Variable} \\
T & ::= & \mathsf{Int} \mid \mathsf{Float} \mid \mathsf{Bool} \mid & \\
 & & \mathsf{Edge} \mid \mathsf{Vertex} & \text{Type}
\end{array}$$

(b) Grammar

```
def Synth$\mathcal{I}$ ($\mathcal{F}$)
$I_1$     memoize the variable $v$ for type Vertex and size 1
$I_2$     foreach (literal $l_i$ with type $T_i$)
$I_3$         memoize $l_i$ for $T_i$ and size 1
$I_4$     $size \leftarrow 1$
$I_5$     while (true)
$I_6$         $E \leftarrow$ Candidates(return type of $\mathcal{F}$, $size$)
$I_7$         foreach ($e \in E$)
$I_8$             if $\mathcal{F}; \Gamma \vdash (\mathbb{C}_1 \wedge \mathbb{C}_2)[\mathcal{I} \mapsto (\lambda v.\ e)]$
$I_9$                 return ($\lambda v.\ e$)
$I_{10}$        $size \leftarrow size + 1$
```

(c) Synthesis of the initialization function $\mathcal{I}$

```
def Synth$\mathcal{P}$ ($\mathcal{F}$, $\mathcal{R}$)
          let $T$ be the return type of $\mathcal{F}$.
$P_1$     memoize variable $n$ for $T$ and size 1
$P_2$     memoize variable $l$ for type Edge and size 1
$P_3$     foreach (literal $l_i$ with type $T_i$)
$P_4$         memoize $l_i$ for $T_i$ and size 1
$P_5$     $size \leftarrow 1$
$P_6$     while (true)
$P_7$         $E \leftarrow$ Candidates ($T$, $size$)
$P_8$         foreach ($e \in E$)
$P_9$             if $\mathcal{F}; \mathcal{R}; \Gamma \vdash (\mathbb{C}_4 \wedge \mathbb{C}_5)[\mathcal{P} := (\lambda n, l.\ e)]$
$P_{10}$                return ($\lambda n, l.\ e$)
$P_{11}$        $size \leftarrow size + 1$
```

(d) Synthesis of the propagation function $\mathcal{P}$

Fig. 20. Synthesis Grammar and functions

```
Γ =
P := List[V],

length: P ⟶ ℕ
elength: ⟨V, V⟩ ⟶ ℕ
∀⟨u, v⟩. if (u = v)  elength(⟨u, v⟩) = 0
    else
        elength(⟨u, v⟩) = 1
∀p. if (p = ⊥)  length(p) = 0
    else
        let v := head(p), p′ := tail(p) in
        if (p′ = ⊥)  length(p) = 0
        else
            let v′ := head(p′) in
            length(p) =
              length(p′) + elength(⟨v′, v⟩)

penultimate: P ⟶ V
∀p. if (p = ⊥)  penultimate(p) = ⊥
    else
        let v := head(p), p′ := tail(p) in
        if (p′ = ⊥)  penultimate(p) = v
        else
            penultimate(p) = head(p′)

weight: ⟨V, V⟩ ⟶ ℕ
∀v. eweight(⟨v, v⟩) = 0
∀p. if (p = ⊥)  weight(p) = 0
    else
        let v := head(p), p′ := tail(p) in
        if (p′ = ⊥)  weight(p) = 0
        else
            let v′ := head(p′) in
            weight(p) =
              weight(p′) + eweight(⟨v′, v⟩)

capacity: P ⟶ ℕ
ecapacity: ⟨V, V⟩ ⟶ ℕ
∀v. ecapacity(⟨v, v⟩) = ⊥
∀p. if (p = ⊥)  capacity(p) = ⊥
    else
        let v := head(p), p′ := tail(p) in
        if (p′ = ⊥)  capacity(p) = ⊥
        else
            let v′ := head(p′) in
            capacity(p) =
              min(capacity(p′), ecapacity(⟨v′, v⟩))
```

Fig. 21. Context assertions Γ

4 Proofs

4.1 Helper Definitions

Definition 19 (Substitution).

Substitution $E := N$:

$N := \mathsf{n} \mid \langle N, N \rangle$

$\langle E, E' \rangle[X := N] = \langle E[X := N], E'[X := N] \rangle$

$e[\langle X, X' \rangle := \langle N, N' \rangle] = e[X := N][X' := N']$

$e \oplus e'[x := \mathsf{n}] = e[x := \mathsf{n}] \oplus e'[x := \mathsf{n}]$

$x[x := \mathsf{n}] = \mathsf{n}$

$x'[x := \mathsf{n}] = x'$

The definitions of substitution for $e := D$, $E := D$, *and* $R := D$ *are similar.*

$D := d \mid \langle D, D \rangle$

4.2 Semantics Compositionality

Lemma 2 (Compositionality for r).
For all r, r' *and* $\mathbb{R}$, *if* $[\![\, r \,]\!] = [\![\, r' \,]\!]$ *then* $[\![\, \mathbb{R}[r] \,]\!] = [\![\, \mathbb{R}[r'] \,]\!]$.

Proof.
Induction on $\mathbb{R}$:
Case
(1) $\mathbb{R} = [\ ]$
Immediate.
Case
(2) $\mathbb{R} = \mathbb{R}' \oplus r$
Immediate by the rule SRBin.

Lemma 3 (Compositionality for m).
For all m, m', *and* $\mathbb{M}$, *if* $[\![\, m \,]\!] = [\![\, m' \,]\!]$ *then* $[\![\, \mathbb{M}[m] \,]\!] = [\![\, \mathbb{M}[m'] \,]\!]$.

Proof.
Induction on $\mathbb{M}$:
Case
(1) $\mathbb{M} = [\ ]$
Immediate.
Case
(2) $\mathbb{M} = \underset{V}{\mathcal{R}}\ \mathbb{M}'$
Immediate by the rule SVRed.
Case
(3) $\mathbb{M} = \mathbb{M}' \oplus m$
Immediate by the rule SMBin.
Case
(4) $\mathbb{M} = m \oplus \mathbb{M}'$
Immediate by the rule SMBin.

Lemma 4 (Compositionality for M).
For all M, M', *and* $\mathbb{M}\mathrm{s}$, *if* $[\![\, M \,]\!] = [\![\, M' \,]\!]$ *then* $[\![\, \mathbb{M}\mathrm{s}[M] \,]\!] = [\![\, \mathbb{M}\mathrm{s}[M'] \,]\!]$.

Proof.
Induction on $\mathbb{M}\mathrm{s}$:
Case
(1) $\mathbb{M}\mathrm{s} = [\ ]$
Immediate.
Case
(2) $\mathbb{M}\mathrm{s} = \langle \mathbb{M}\mathrm{s}, M \rangle$
Immediate by the rule SMPair.
Case
(3) $\mathbb{M}\mathrm{s} = \langle M, \mathbb{M}\mathrm{s} \rangle$
Immediate by the rule SMPair.
Case
(4) $\mathbb{M}\mathrm{s} = \mathrm{ilet}\ X := \mathbb{M}\mathrm{s}\ \mathrm{in}\ e$
Immediate by the rule SMLet.

Case

(5) $\mathbb{M}\text{s} =$

$\text{ilet } X := \mathbb{M}\text{s in}$
$\text{mlet } X := E \text{ in}$
$\text{rlet } X := R \text{ in } e$

Immediate by the rule SRLet.

Lemma 5 (Compositionality for R).
For all R, R', and $\mathbb{R}\text{s}$, if $\llbracket R \rrbracket = \llbracket R' \rrbracket$ then $\llbracket \mathbb{R}\text{s}[R] \rrbracket = \llbracket \mathbb{R}\text{s}[R'] \rrbracket$.

Proof.
Induction on $\mathbb{R}\text{s}$:

Case

(1) $\mathbb{R}\text{s} = [\,]$

Immediate.

Case

(2) $\mathbb{R}\text{s} = \langle \mathbb{R}\text{s}, R \rangle$

Immediate by the rule SRPair.

Case

(3) $\mathbb{R}\text{s} = \langle R, \mathbb{R}\text{s} \rangle$

Immediate by the rule SRPair.

Case

(4) $\mathbb{R}\text{s} =$

$\text{ilet } X := M \text{ in}$
$\text{mlet } X := E \text{ in}$
$\text{rlet } X := \mathbb{R}\text{s in } e$

Immediate by the rule SRLet.

4.3 Soundness of Fusion

Theorem 21 (Semantics-preserving Fusion for r).
For all r_1 and r_2, if $r_1 \Rightarrow_r r_2$ then $[\![r_1]\!] = [\![r_2]\!]$.

Proof.
Case analysis on $r_1 \Rightarrow_r r_2$:

Case rule FMInR:
Immediate from Lemma 6 and Lemma 4.

Case rule FVRed:
Immediate the rules SVRed and SMLet on r_1 and SRLet on r_2.

Case rule FLetsBin:
By the rules SRLet, SMPair, SEEPair, SRPair, SEBin.
Similar to Lemma 6, the case for FILetBin.

Case rule FMInLets:
Immediate from Lemma 7, Lemma 4 and SRLet.

Case rule FRInLets:
Immediate from Lemma 8, Lemma 5 and SRLet.

Lemma 6 (Semantics-preserving Fusion for m).
For all m_1 and m_2, if $m_1 \Rightarrow_m m_2$ then $[\![m_1]\!] = [\![m_2]\!]$.

Proof.
Induction on $m_1 \Rightarrow_m m_2$:

Case rule FMInM:
Immediate from the induction hypothesis and Lemma 3.

Case rule FPNest:

(1) $s = \underset{\substack{p \in \text{args}\, \mathcal{R}' \; \mathcal{F}'(p') \\ p' \in P}}{\mathcal{R}} \mathcal{F}(p)$

(2) $s' = \text{ilet } \langle x, x' \rangle := \underset{p' \in P}{\mathcal{R}''} \mathcal{F}''(p') \text{ in } x'$

(3) $\mathcal{R}' \in \{\min, \max\}$

(4) $f'' := \lambda p.\ \langle \mathcal{F}'(p), \mathcal{F}(p) \rangle$

(5) $\mathcal{R}''(\langle a, b \rangle, \langle a', b' \rangle) :=$
$\quad \text{if } (a' = a) \text{ then } \langle a, \mathcal{R}(b, b') \rangle$
$\quad \text{else if } (\mathcal{R}'(a, a') = a) \text{ then } \langle a, b \rangle \text{ else } \langle a', b' \rangle$

By the rules SPRed and SArgsR on [1],

$$(6)\ \ [\![s]\!] = \overline{\left[\text{v} \mapsto \mathcal{R}\ \left\{ \mathcal{F}(p) \mid p \in \left\{ p \mid p \in \{\overline{p}\} \wedge \mathcal{F}'(p) = \mathcal{R}'\ \left\{ \mathcal{F}'(p) \mid p' \in \{\overline{p}\} \right\} \right\} \right\} \right]}_{\text{v} \in \text{V}(g)}$$

where

(7) $\{\overline{p}\} = [\![P]\!](g)(\text{v})$

(8) $\mathcal{R}' \in \{\min, \max\}$

By the rules SMLet and SPRed on [2],

$$(9)\ \ [\![s']\!] = \overline{[\text{v} \mapsto \text{second}\,(\mathcal{R}''\ \{\mathcal{F}''(p) \mid p \in [\![P]\!](g)(\text{v})\})]}_{\text{v} \in \text{V}(g)}$$

From [7] and [9],

(10) $[\![s']\!] = \overline{\left[\mathsf{v} \mapsto \mathcal{R}''\ \left\{\mathcal{F}''(p) \mid p \in \{\overline{p}\}\right\}\right]}_{\mathsf{v}\in\mathsf{V}(g)}$

From [6] and [10], we need to show that for all P,

(11) $\mathcal{R}\ \{\mathcal{F}(p) \mid p \in P \land \mathcal{F}'(p) = \mathcal{R}'\ \{\mathcal{F}'(p) \mid p' \in P\}\} =$
$\quad \text{second}\,(\mathcal{R}''\ \{\mathcal{F}''(p) \mid p \in P\})$

The proof is by by induction on P.

Base Case:

(12) $P = \{p^*\}$

Form [12],

(13) $\mathcal{R}\ \{\mathcal{F}(p) \mid p \in P \land \mathcal{F}'(p) = \mathcal{R}'\ \{\mathcal{F}'(p) \mid p' \in P\}\} =$
$\quad \mathcal{R}\ \{\mathcal{F}(p) \mid p \in \{p^*\} \land \mathcal{F}'(p) = \mathcal{R}'\ \{\mathcal{F}'(p) \mid p' \in \{p^*\}\}\} =$
$\quad \mathcal{R}\ \{\mathcal{F}(p) \mid p \in \{p^*\} \land \mathcal{F}'(p) = \mathcal{F}'(p^*)\} =$
$\quad \mathcal{F}(p^*)$

Form [12] and [4],

(14) $\text{second}\,(\mathcal{R}''\ \{\mathcal{F}''(p) \mid p \in P\}) =$
$\quad \text{second}\,(\langle \mathcal{F}'(p^*), \mathcal{F}(p^*)\rangle) =$
$\quad \mathcal{F}(p^*)$

The conclusion is immediate from [13] and [14],

Inductive Case:

(15) $P = P' \cup \{p^*\}$

Induction Hypothesis:

(16) $\mathcal{R}\ \{\mathcal{F}(p) \mid p \in P' \land \mathcal{F}'(p) = \mathcal{R}'\ \{\mathcal{F}'(p) \mid p' \in P'\}\} =$
$\quad \text{second}\,(\mathcal{R}''\ \{\mathcal{F}''(p) \mid p \in P'\})$

We show that

$\mathcal{R}\ \{\mathcal{F}(p) \mid p \in P' \cup \{p^*\} \land \mathcal{F}'(p) = \mathcal{R}'\ \{\mathcal{F}'(p) \mid p' \in P' \cup \{p^*\}\}\} =$
$\quad \text{second}\,(\mathcal{R}''\ \{\mathcal{F}''(p) \mid p \in P' \cup \{p^*\}\})$

That is

$\mathcal{R}\ \{\mathcal{F}(p) \mid p \in P' \cup \{p^*\} \land \mathcal{F}'(p) = \mathcal{R}'\,(\mathcal{R}'\ \{\mathcal{F}'(p) \mid p' \in P'\}, \mathcal{F}'(p^*))\} =$
$\quad \text{second}\,(\mathcal{R}''\,(\mathcal{R}''\ \{\mathcal{F}''(p) \mid p \in P'\}, \mathcal{F}''(p^*)))$

From [5], By induction on S, it can be proved that

(17) $\forall S.\ \text{first}(\mathcal{R}''\ S) = \mathcal{R}'\ \{a \mid \langle a, a'\rangle \in S\}$

We consider two cases:

Case

(18) $\mathcal{F}'(p^*) = \mathcal{R}'\ \{\mathcal{F}'(p) \mid p' \in P'\}$

From [18],

(19) $\mathcal{R}\ \{\mathcal{F}(p) \mid p \in P' \cup \{p^*\} \land \mathcal{F}'(p) = \mathcal{R}'\,(\mathcal{R}'\ \{\mathcal{F}'(p) \mid p' \in P'\}, \mathcal{F}'(p^*))\} =$
$\quad \mathcal{R}\ \{\mathcal{F}(p) \mid p \in P' \cup \{p^*\} \land \mathcal{F}'(p) = \mathcal{F}'(p^*) = \{\mathcal{F}'(p) \mid p' \in P'\}\} =$
$\quad \mathcal{R}\,(\mathcal{R}\ \{\mathcal{F}(p) \mid p \in P' \cup \{p^*\} \land \mathcal{F}'(p) = \{\mathcal{F}'(p) \mid p' \in P'\}\}, \mathcal{F}(p^*))$

From [18] and [17],

(20) $\text{first}(\mathcal{R}''\ \{\mathcal{F}''(p) \mid p \in P'\}) = \mathcal{F}'(p^*)$

We have

(21) $\text{second}\,(\mathcal{R}''\,(\mathcal{R}''\ \{\mathcal{F}''(p) \mid p \in P'\}, \mathcal{F}''(p^*))) =$

By [4],

$\quad \text{second}\,(\mathcal{R}''\,(\mathcal{R}''\ \{\mathcal{F}''(p) \mid p \in P'\}, \langle \mathcal{F}'(p^*), \mathcal{F}(p^*)\rangle)) =$

By [5] and [20],

$\quad \text{second}\,(\langle \mathcal{F}'(p^*), \mathcal{R}\,(\text{second}\,(\mathcal{R}''\ \{\mathcal{F}''(p) \mid p \in P'\}), \mathcal{F}(p^*))\rangle) =$
$\quad \mathcal{R}\,(\text{second}\,(\mathcal{R}''\ \{\mathcal{F}''(p) \mid p \in P'\}), \mathcal{F}(p^*))$

Thus

(22) $\text{second}\,(\mathcal{R}''\,(\mathcal{R}''\ \{\mathcal{F}''(p) \mid p \in P'\}\,, \mathcal{F}''(p^*))) =$

$\mathcal{R}\,(\text{second}\,(\mathcal{R}''\ \{\mathcal{F}''(p) \mid p \in P'\})\,, \mathcal{F}(p^*))$

From [19] and [22], we have the conclusion:

$\mathcal{R}\ \{\mathcal{F}(p) \mid p \in P' \cup \{p^*\} \wedge \mathcal{F}'(p) = \mathcal{R}'\,(\mathcal{R}'\ \{\mathcal{F}'(p) \mid p' \in P'\}\,, \mathcal{F}'(p^*))\} =$

$\text{second}\,(\mathcal{R}''\,(\mathcal{R}''\ \{\mathcal{F}''(p) \mid p \in P'\}\,, \mathcal{F}''(p^*)))$

Case

(23) $\mathcal{F}'(p^*) \neq \mathcal{R}'\ \{\mathcal{F}'(p) \mid p' \in P'\}$

We assume $\mathcal{R}' = \max$. The other case $\mathcal{R}' = \min$ is similar.

We consider two sub-cases.

Sub-case

(24) $\mathcal{R}'\,(\mathcal{R}'\ \{\mathcal{F}'(p) \mid p' \in P'\}\,, \mathcal{F}'(p^*)) = \mathcal{F}'(p^*)$

From [23] and [24],

(25) $\forall p' \in P'.\ \mathcal{F}'(p') < \mathcal{F}'(p^*)$

We have

(26) $\mathcal{R}\ \{\mathcal{F}(p) \mid p \in P' \cup \{p^*\} \wedge \mathcal{F}'(p) = \mathcal{R}'\,(\mathcal{R}'\ \{\mathcal{F}'(p) \mid p' \in P'\}\,, \mathcal{F}'(p^*))\} =$

By [24]

$\mathcal{R}\ \{\mathcal{F}(p) \mid p \in P' \cup \{p^*\} \wedge \mathcal{F}'(p) = \mathcal{F}'(p^*)\} =$

By [25]

$\mathcal{F}(p^*)$

Thus

(27) $\mathcal{R}\ \{\mathcal{F}(p) \mid p \in P' \cup \{p^*\} \wedge \mathcal{F}'(p) = \mathcal{R}'\,(\mathcal{R}'\ \{\mathcal{F}'(p) \mid p' \in P'\}\,, \mathcal{F}'(p^*))\} =$
$\mathcal{F}(p^*)$

From [18] and [23],

(28) $\text{first}(\mathcal{R}''\ \{\mathcal{F}''(p) \mid p \in P'\}) \neq \mathcal{F}'(p^*)$

We have

(29) $\text{second}\,(\mathcal{R}''\,(\mathcal{R}''\ \{\mathcal{F}''(p) \mid p \in P'\}\,, \mathcal{F}''(p^*))) =$

By [4],

$\text{second}\,(\mathcal{R}''\,(\mathcal{R}''\ \{\mathcal{F}''(p) \mid p \in P'\}\,, \langle \mathcal{F}'(p^*), \mathcal{F}(p^*)\rangle)) =$

By [5], [28] and [24],

$\text{second}\,(\langle \mathcal{F}'(p^*), \mathcal{F}(p^*)\rangle) =$

$\mathcal{F}(p^*)$

Thus

(30) $\text{second}\,(\mathcal{R}''\,(\mathcal{R}''\ \{\mathcal{F}''(p) \mid p \in P'\}\,, \mathcal{F}''(p^*))) =$

$\mathcal{F}(p^*)$

From [27] and [30], we have the conclusion:

$\mathcal{R}\ \{\mathcal{F}(p) \mid p \in P' \cup \{p^*\} \wedge \mathcal{F}'(p) = \mathcal{R}'\,(\mathcal{R}'\ \{\mathcal{F}'(p) \mid p' \in P'\}\,, \mathcal{F}'(p^*))\} =$

$\text{second}\,(\mathcal{R}''\,(\mathcal{R}''\ \{\mathcal{F}''(p) \mid p \in P'\}\,, \mathcal{F}''(p^*)))$

Sub-case

(31) $\mathcal{R}'\,(\mathcal{R}'\ \{\mathcal{F}'(p) \mid p' \in P'\}\,, \mathcal{F}'(p^*)) = \mathcal{R}'\ \{\mathcal{F}'(p) \mid p' \in P'\}$

This sub-case is similar to the previous sub-case.

$\mathcal{F}'(p^*)$ and $\mathcal{R}'\ \{\mathcal{F}'(p) \mid p' \in P'\}$ replace each other.

Case rule FPRed:

Immediate from the rules SMLet and SMM.

Case rule FILetBin:

(32) $s = (\text{ilet}\ X_1 := M_1\ \text{in}\ e_1)\ \oplus\ (\text{ilet}\ X_2 := M_2\ \text{in}\ e_2)$

(33) $s' = \mathsf{ilet}\ \langle X_1, X_2\rangle := \langle M_1, M_2\rangle\ \mathsf{in}\ e_1 \oplus e_2$
(34) $\mathsf{free}(e_1) \cap X_2 = \emptyset$
(35) $\mathsf{free}(e_2) \cap X_1 = \emptyset$
By rules SMBin and SMLet on [32], we have
(36) $\llbracket s \rrbracket = \overline{[\mathsf{v} \mapsto \llbracket e_1\, [X_1 := \llbracket M_1 \rrbracket\,(g)(\mathsf{v})]\, \rrbracket \oplus \llbracket e_2\, [X_2 := \llbracket M_2 \rrbracket\,(g)(\mathsf{v})]\, \rrbracket]}_{\mathsf{v}\in\mathsf{V}(g)}$
By rules SMLet on [33], we have
(37) $\llbracket s' \rrbracket = \overline{[\mathsf{v} \mapsto \llbracket (e_1 \oplus e_2)\, [\langle X_1, X_2\rangle := \llbracket \langle M_1, M_2\rangle \rrbracket\,(g)(\mathsf{v})]\, \rrbracket]}_{\mathsf{v}\in\mathsf{V}(g)}$
By [37] and the rule SMPair, we have
(38) $\llbracket s' \rrbracket = \overline{[\mathsf{v} \mapsto \llbracket (e_1 \oplus e_2)\, [\langle X_1, X_2\rangle := \langle \llbracket M_1 \rrbracket\,(g)(\mathsf{v}), \llbracket M_2 \rrbracket\,(g)(\mathsf{v})\rangle]\, \rrbracket]}_{\mathsf{v}\in\mathsf{V}(g)}$
From [38], and the rule SEBin, we have
(39) $\llbracket s' \rrbracket = \overline{[\mathsf{v} \mapsto \llbracket e_1\, [\langle X_1, X_2\rangle := \langle \llbracket M_1 \rrbracket\,(g)(\mathsf{v}), \llbracket M_2 \rrbracket\,(g)(\mathsf{v})\rangle]\, \rrbracket \oplus}$
$\overline{\llbracket e_2\, [\langle X_1, X_2\rangle := \langle \llbracket M_1 \rrbracket\,(g)(\mathsf{v}), \llbracket M_2 \rrbracket\,(g)(\mathsf{v})\rangle]\, \rrbracket]}_{\mathsf{v}\in\mathsf{V}(g)}$
From [39], [34] and [35], we have
(40) $\llbracket s' \rrbracket = \overline{[\mathsf{v} \mapsto \llbracket e_1\, [X_1 := \llbracket M_1 \rrbracket\,(g)(\mathsf{v})]\, \rrbracket \oplus \llbracket e_2\, [X_2 := \llbracket M_2 \rrbracket\,(g)(\mathsf{v})]\, \rrbracket]}_{\mathsf{v}\in\mathsf{V}(g)}$
From [36] and [40], we have
$\llbracket s \rrbracket = \llbracket s' \rrbracket$

Case rule FMIniLet:
Immediate from Lemma 7, Lemma 4 and SMLet.

Lemma 7 (Semantics-preserving Fusion for M).
For all M_1 and M_2, if $M_1 \Rightarrow_M M_2$ then $\llbracket M_1 \rrbracket = \llbracket M_2 \rrbracket$.

Proof.
Induction on $M_1 \Rightarrow_M M_2$:
Case rule FMPair:
(1) $M_1 = \langle \mathcal{R}\ \mathcal{F}, \mathcal{R}'\ \mathcal{F}'\rangle$
(2) $M_2 = \mathcal{R}''\ F''$
(3) $f'' := \lambda p.\ \langle \mathcal{F}'(p), \mathcal{F}(p)\rangle$
(4) $\mathcal{R}''(\langle a, b\rangle, \langle a', b'\rangle) := \langle \mathcal{R}(a, a'), \mathcal{R}'(b, b')\rangle$
By SMPair, SMM and SPRed on [1], we have
(5) $\llbracket M_1 \rrbracket = \left\langle \overline{[\mathsf{v} \mapsto \mathcal{R}\ \{\mathcal{F}(p) \mid p \in \llbracket \mathsf{Paths} \rrbracket\,(\mathsf{v})\}]}_{\mathsf{v}\in\mathsf{V}(g)}, \overline{[\mathsf{v} \mapsto \mathcal{R}'\ \{\mathcal{F}'(p) \mid p \in \llbracket \mathsf{Paths} \rrbracket\,(\mathsf{v})\}]}_{\mathsf{v}\in\mathsf{V}(g)} \right.$
By SMM and SPRed on [2] and [3] and [4], we have
(6) $\llbracket M_2 \rrbracket = \overline{[\mathsf{v} \mapsto \mathcal{R}''\ \{\langle \mathcal{F}(p), \mathcal{F}'(p)\rangle \mid p \in \llbracket \mathsf{Paths} \rrbracket\,(\mathsf{v})\}]}_{\mathsf{v}\in\mathsf{V}(g)}$
By [6] and [4], we have
(7) $\llbracket M_2 \rrbracket = \left\langle \overline{[\mathsf{v} \mapsto \mathcal{R}\ \{\mathcal{F}(p) \mid p \in \llbracket \mathsf{Paths} \rrbracket\,(\mathsf{v})\}]}_{\mathsf{v}\in\mathsf{V}(g)}, \overline{[\mathsf{v} \mapsto \mathcal{R}'\ \{\mathcal{F}'(p) \mid p \in \llbracket \mathsf{Paths} \rrbracket\,(\mathsf{v})\}]}_{\mathsf{v}\in\mathsf{V}(g)} \right.$
From [5] and [7], we have
(8) $\llbracket M_1 \rrbracket = \llbracket M_2 \rrbracket$

Lemma 8 (Semantics-preserving Fusion for R).
For all R_1, R_2, X and $\overline{d}$ where $d \in \mathcal{D}_m$, if $R_1 \Rightarrow_R R_2$ then $\left\llbracket R_1 [X := \overline{d}] \right\rrbracket = \left\llbracket R_2 [X := \overline{d}] \right\rrbracket$.

Proof.
Induction on $R_1 \Rightarrow_R R_2$:
Case rule FRPair:

(1) $R_1 = \langle \mathcal{R}_1\ x_1, \mathcal{R}_2\ x_2 \rangle$

(2) $R_2 = \mathcal{R}_3\ \langle x_1, x_2 \rangle$

(3) $\mathcal{R}_3(\langle a, b\rangle, \langle a', b'\rangle) := \langle \mathcal{R}_1(a, a'), \mathcal{R}_2(b, b')\rangle$

If x_1 or $x_2 \notin X$, $x_1[X := \overline{d}] = \bot$ or $x_2[X := \overline{d}] = \bot$
as the rule SRR is the only semantic rule for R,

$$(4)\quad \left[\!\left[R_1[X := \overline{d}] \right]\!\right] = \left[\!\left[R_2[X := \overline{d}] \right]\!\right] = \bot.$$

Thus, the remained case is that

$$(5)\quad x_1 := \overline{[\mathsf{v} \mapsto n_{\mathsf{v}}]}_{\mathsf{v}\in\mathsf{V}(g)} \in X := \overline{d}$$

$$(6)\quad x_2 := \overline{[\mathsf{v} \mapsto n'_{\mathsf{v}}]}_{\mathsf{v}\in\mathsf{V}(g)} \in X := \overline{d}$$

From [1], [5] and [6], we have

$$(7)\quad R_1 = \left\langle \mathcal{R}_1\ \overline{[\mathsf{v} \mapsto n_{\mathsf{v}}]}_{\mathsf{v}\in\mathsf{V}(g)}, \mathcal{R}_2\ \overline{[\mathsf{v} \mapsto n'_{\mathsf{v}}]}_{\mathsf{v}\in\mathsf{V}(g)} \right\rangle$$

From [2], [5] and [6], we have

$$(8)\quad R_2 = \mathcal{R}_3\ \left\langle \overline{[\mathsf{v} \mapsto n_{\mathsf{v}}]}_{\mathsf{v}\in\mathsf{V}(g)}, \overline{[\mathsf{v} \mapsto n'_{\mathsf{v}}]}_{\mathsf{v}\in\mathsf{V}(g)} \right\rangle$$

By SRPair, SRR and SVRed on [7], we have

$$(9)\quad [\![R_1]\!] = \left\langle \mathcal{R}_1\ \{\overline{n_{\mathsf{v}}}\ _{\mathsf{v}\in\mathsf{V}(g)}\}, \mathcal{R}_2\ \{\overline{n'_{\mathsf{v}}}\ _{\mathsf{v}\in\mathsf{V}(g)}\} \right\rangle$$

By SRPair, SRR and SVRed on [8], we have

$$(10)\quad [\![R_2]\!] = \mathcal{R}_3\ \left\{ \overline{\langle n_{\mathsf{v}}, n'_{\mathsf{v}}\rangle}_{\mathsf{v}\in\mathsf{V}(g)} \right\}$$

From [10] and [3], we have

$$(11)\quad [\![R_2]\!] = \langle \mathcal{R}_1\ \{\overline{n_{\mathsf{v}}}\}_{\mathsf{v}\in\mathsf{V}(g)}, \mathcal{R}_2\ \{\overline{n'_{\mathsf{v}}}\}_{\mathsf{v}\in\mathsf{V}(g)} \rangle$$

From [9] and [11], we have

$$\left[\!\left[R_1[X := \overline{d}] \right]\!\right] = \left[\!\left[R_2[X := \overline{d}] \right]\!\right].$$

4.4 Iteration Correctness Conditions

4.4.1 Pull, Idempotent

Theorem 22 (Correctness of Pull (idempotent reduction)). *For all $\mathcal{R}$, $\mathcal{F}$, C, $\mathcal{I}$, $\mathcal{P}$, and $k \geq 1$, if the conditions $\mathbb{C}_1$ - $\mathbb{C}_9$ hold,*
$\mathcal{S}^k_{\text{pull+}}(v) = \mathcal{S}pec^k(v)$

We assume that

(1) $\forall n.\ \mathcal{R}(n, \bot) = n$
(2) $\forall n, n'.\ \mathcal{R}(n, n') = \mathcal{R}(n', n)$
(3) $\forall n, n', n''.\ \mathcal{R}(\mathcal{R}(n, n'), n'') = \mathcal{R}(n, \mathcal{R}(n', n''))$
(4) $\forall n.\ \mathcal{R}(n, n) = n$
(5) $\forall v \in \mathsf{V}.\ C(\langle v, v \rangle) \longrightarrow \mathcal{I}(v) = \mathcal{F}(\langle v, v \rangle)$
(6) $\forall v \in \mathsf{V}.\ \neg C(\langle v, v \rangle) \longrightarrow \mathcal{I}(v) = \bot$
(7) $\forall e.\ \mathcal{P}(\bot, e) = \bot$
(8) $\forall p_1,\ p_2 \in \mathsf{P},\ v \in \mathsf{V}.$
$\mathsf{tail}(p_1) = \mathsf{tail}(p_2) \longrightarrow$
$\mathsf{let}\ u := \mathsf{tail}(p_1)\ \mathsf{in}$
$\mathcal{P}\left[\mathcal{R}(\mathcal{F}(p_1), \mathcal{F}(p_2)),\ \langle u, v \rangle\right] =$
$\mathcal{R}\left[\mathcal{F}(p_1 \cdot \langle u, v \rangle),\ \mathcal{F}(p_2 \cdot \langle u, v \rangle)\right]$
(9) $\forall p, e.\ \mathcal{P}(\mathcal{F}(p),\ e) = \mathcal{F}(p \cdot e)$

Form Def. 6, we have

$$\mathcal{S}pec^k(v) = \mathcal{R}_{p \,\in\, \{p \,|\, p \in \mathsf{Paths}(v)\ \wedge\ C(p)\ \wedge\ \mathsf{length}(p) < k\}}\ \mathcal{F}(p)$$

Proof by induction on k:

Base Case:

$k = 1$

We should show that

$$\mathcal{S}^1_{\text{pull+}}(v) = \mathcal{R}_{p \,\in\, \{p \ |\ p \in \mathsf{Paths}(v)\ \wedge\ C(p)\ \wedge\ \mathsf{length}(p) < 1\}} \mathcal{F}(p)$$

that is

$$\mathcal{S}^1_{\text{pull+}}(v) = \mathcal{R}_{p \,\in\, \{p \ |\ p = \langle v, v \rangle\ \wedge\ C(p)\}} \mathcal{F}(p)$$

We consider two cases:

Case:

(10) $C(\langle v, v \rangle)$

We should show that

$$\mathcal{S}^1_{\text{pull+}}(v) = \mathcal{R}_{\{\langle v, v \rangle\}} \mathcal{F}(p)$$

that is

$$\mathcal{S}^1_{\text{pull+}}(v) = \mathcal{F}(\langle v, v \rangle)$$

that is straightforward from Def. 1, [5] and [10].

Case:

(11) $\neg C(\langle v, v \rangle)$

We should show that

$$\mathcal{S}^1_{\text{pull+}}(v) = \underset{\emptyset}{\mathcal{R}}\, \mathcal{F}(p)$$

that is

$$\mathcal{S}^1_{\text{pull+}}(v) = \bot$$

that is straightforward from Def. 1, [6] and [11].

Inductive Case:

(12) $k > 1$

The induction hypothesis is:

(13) $\mathcal{S}^{k'}_{\text{pull+}}(v) = \mathcal{R}_{p \in \{p \mid p \in \text{Paths}(v) \wedge C(p) \wedge \text{length}(p) < k'\}} \mathcal{F}(p)$ for all v and $k' \leq k$

We should show that

$\mathcal{S}^{k+1}_{\text{pull+}}(v) = \mathcal{R}_{p \in \{p \mid p \in \text{Paths}(v) \wedge C(p) \wedge \text{length}(p) < k+1\}} \mathcal{F}(p)$

From Def. 1, we consider two cases:

Case:

(14) $\text{CPreds}^{k}(v) \neq \emptyset$

(15) $\mathcal{S}^{k+1}_{\text{pull+}}(v) = \mathcal{R}\left[\mathcal{S}^{k}_{\text{pull+}}(v),\ \mathcal{R}_{u \in \text{preds}(v)}\ \mathcal{P}\left(\mathcal{S}^{k}_{\text{pull+}}(u), \langle u, v\rangle\right)\right]$

From [15] and [13]

(16) $\mathcal{S}^{k+1}_{\text{pull+}}(v) = \mathcal{R}[$

$\mathcal{S}^{k}_{\text{pull+}}(v),$

$\mathcal{R}_{u \in \text{preds}(v)}\ \left[\mathcal{P}\ \left(\mathcal{R}_{p \in \{p \mid p \in \text{Paths}(u) \wedge C(p) \wedge \text{length}(p) < k\}} \mathcal{F}(p),\ \langle u, v\rangle\right)\right]]$

In the case that the size of the set of paths is more than one, from [8], and in the case that the set of paths is singleton, from $\mathcal{R}_{\{v\}} = v$ and [9], and in the case that the set of paths is empty, from $\mathcal{R}_{\emptyset} = \bot$ and [7],

we have

(17) $\mathcal{P}\ \left(\mathcal{R}_{p \in \{p \mid p \in \text{Paths}(u) \wedge C(p) \wedge \text{length}(p) < k\}} \mathcal{F}(p),\ \langle u, v\rangle\right) =$

$\mathcal{R}_{p \in \{p \mid p \in \text{Paths}(u) \wedge C(p) \wedge \text{length}(p) < k\}} \mathcal{F}(p \cdot \langle u, v\rangle)$

After substituting [17] in [16]

(18) $\mathcal{S}^{k+1}_{\text{pull+}}(v) = \mathcal{R}[$

$\mathcal{S}^{k}_{\text{pull+}}(v),$

$\mathcal{R}_{u \in \text{preds}(v)}\ \left[\mathcal{R}_{p \in \{p \mid p \in \text{Paths}(u) \wedge C(p) \wedge \text{length}(p) < k\}} \mathcal{F}(p \cdot \langle u, v\rangle)\right]]$

that is

(19) $\mathcal{S}^{k+1}_{\text{pull+}}(v) = \mathcal{R}[$

$\mathcal{S}^{k}_{\text{pull+}}(v),$

$\mathcal{R}_{p \in \{p \mid \exists u.\ p \in \text{Paths}(u) \wedge u \in \text{preds}(v) \wedge C(p) \wedge \text{length}(p) < k\}} \mathcal{F}(p \cdot \langle u, v\rangle)]$

From [19] and Lemma 9

(20) $\mathcal{S}^{k+1}_{\text{pull+}}(v) = \mathcal{R}[$

$\mathcal{S}^{k}_{\text{pull+}}(v),$

$\mathcal{R}_{p \in \{p \mid \exists u.\ p \in \text{Paths}(u) \wedge u \in \text{preds}(v) \wedge C(p \cdot \langle u, v\rangle) \wedge \text{length}(p) < k\}} \mathcal{F}(p \cdot \langle u, v\rangle)]$

that is

(21) $\mathcal{S}^{k+1}_{\text{pull+}}(v) = \mathcal{R}[$

$\mathcal{S}^{k}_{\text{pull+}}(v),$

$\mathcal{R}_{p' \in \{p' \mid p' \in \text{Paths}(v) \wedge C(p') \wedge 0 < \text{length}(p') < k+1\}} \mathcal{F}(p')]$

From [21] and [13]

(22) $\mathcal{S}^{k+1}_{\text{push+}}(v) = \mathcal{R}[$

$\mathcal{R}_{p \in \{p \mid p \in \text{Paths}(v) \wedge C(p) \wedge \text{length}(p) < k\}} \mathcal{F}(p),$

$\mathcal{R}_{p \in \{p \mid p \in \text{Paths}(v) \wedge C(p) \wedge 0 < \text{length}(p) < k+1\}} \mathcal{F}(p)]$

From [22] and [4]

$\mathcal{S}^{k+1}_{\text{push+}}(v) = \mathcal{R}_{p \in \{p \mid p \in \text{Paths}(v) \wedge C(p) \wedge \text{length}(p) < k+1\}} \mathcal{F}(p),$

Case:

(23) $\mathsf{CPreds}^k(v) = \emptyset$

(24) $\mathcal{S}^{k+1}_{\mathsf{pull+}}(v) = \mathcal{S}^{k}_{\mathsf{pull+}}(v)$

(25) $\mathsf{CPreds}^k(v) = \left\{u \mid u \in \mathsf{preds}(v) \wedge \mathcal{S}^{k}_{\mathsf{pull+}}(u) \neq \mathcal{S}^{k-1}_{\mathsf{pull+}}(u)\right\}$

From [13] and [24],

(26) $\mathcal{S}^{k+1}_{\mathsf{pull+}}(v) = \mathcal{R}_{p \,\in\, \{p \;\mid\; p \in \mathsf{Paths}(v) \,\wedge\, C(p) \,\wedge\, \mathsf{length}(p) < k\}} \mathcal{F}(p)$

From [26] and [1],

(27) $\mathcal{S}^{k+1}_{\mathsf{pull+}}(v) = \mathcal{R}[$
$\mathcal{R}_{p \,\in\, \{p \;\mid\; \exists u.\ p \in \mathsf{Paths}(u) \,\wedge\, u \in \mathsf{preds}(v) \,\wedge\, C(p \cdot \langle u,v \rangle) \,\wedge\, 0 < \mathsf{length}(p \cdot \langle u,v \rangle) < k\}} \mathcal{F}(p),$
$\mathcal{R}_{p \,\in\, \{p \;\mid\; p = \langle v,v \rangle \,\wedge\, C(\langle v,v \rangle)\}} \mathcal{F}(p)]$

From [27] and Lemma 9,

(28) $\mathcal{S}^{k+1}_{\mathsf{pull+}}(v) = \mathcal{R}[$
$\mathcal{R}_{p \,\in\, \{p \;\mid\; \exists u.\ p \in \mathsf{Paths}(u) \,\wedge\, u \in \mathsf{preds}(v) \,\wedge\, C(p) \,\wedge\, 0 < \mathsf{length}(p \cdot \langle u,v \rangle) < k\}} \mathcal{F}(p),$
$\mathcal{R}_{p \,\in\, \{p \;\mid\; p = \langle v,v \rangle \,\wedge\, C(\langle v,v \rangle)\}} \mathcal{F}(p)]$

From [28], [2] and [3]

(29) $\mathcal{S}^{k+1}_{\mathsf{pull+}}(v) = \mathcal{R}[$
$\mathcal{R}_{u \in \mathsf{preds}(v)}\ \mathcal{R}_{p' \,\in\, \{p' \;\mid\; \exists u.\ p' \in \mathsf{Paths}(u) \,\wedge\, C(p') \,\wedge\, \mathsf{length}(p') < k-1\}} \mathcal{F}(p' \cdot \langle u, v \rangle),$
$\mathcal{R}_{p \,\in\, \{p \;\mid\; p = \langle v,v \rangle \,\wedge\, C(\langle v,v \rangle)\}} \mathcal{F}(p)]$

In the case that the size of the set of paths is more than one, from [8], and in the case that the set of paths is singleton, from $\mathcal{R}_{\{v\}} = v$ and [9], and in the case that the set of paths is empty, from $\mathcal{R}_{\emptyset} = \bot$ and [7],

(30) $\mathcal{P}\left(\mathcal{R}_{p \,\in\, \{p \;\mid\; p \in \mathsf{Paths}(u) \,\wedge\, C(p) \,\wedge\, \mathsf{length}(p) < k-1\}} \mathcal{F}(p),\ \langle u, v \rangle\right) =$
$\mathcal{R}_{p \,\in\, \{p \;\mid\; p \in \mathsf{Paths}(u) \,\wedge\, C(p) \,\wedge\, \mathsf{length}(p) < k-1\}} \mathcal{F}(p \cdot \langle u, v \rangle)$

From [29] and [30], we have

(31) $\mathcal{S}^{k+1}_{\mathsf{pull+}}(v) = \mathcal{R}[$
$\mathcal{R}_{u \in \mathsf{preds}(v)}\ \mathcal{P}\left(\mathcal{R}_{p \,\in\, \{p \;\mid\; p \in \mathsf{Paths}(u) \,\wedge\, C(p) \,\wedge\, \mathsf{length}(p) < k-1\}} \mathcal{F}(p),\ \langle u, v \rangle\right),$
$\mathcal{R}_{p \,\in\, \{p \;\mid\; p = \langle v,v \rangle \,\wedge\, C(\langle v,v \rangle)\}} \mathcal{F}(p)]$

From [31] and [13], we have

(32) $\mathcal{S}^{k+1}_{\mathsf{pull+}}(v) = \mathcal{R}[$
$\mathcal{R}_{u \in \mathsf{preds}(v)}\ \mathcal{P}\left(\mathcal{S}^{k-1}_{\mathsf{pull+}}(u),\ \langle u, v \rangle\right),$
$\mathcal{R}_{p \,\in\, \{p \;\mid\; p = \langle v,v \rangle \,\wedge\, C(\langle v,v \rangle)\}} \mathcal{F}(p)]$

From [23] and [25], we have

(33) For all $u \in \mathsf{preds}(v)$: $\mathcal{S}^{k}_{\mathsf{pull+}}(u) = \mathcal{S}^{k-1}_{\mathsf{pull+}}(u)$

From [32] and [33], we have

(34) $\mathcal{S}^{k+1}_{\mathsf{pull+}}(v) = \mathcal{R}[$
$\mathcal{R}_{u \in \mathsf{preds}(v)}\ \mathcal{P}\left(\mathcal{S}^{k}_{\mathsf{pull+}}(u),\ \langle u, v \rangle\right),$
$\mathcal{R}_{p \,\in\, \{p \;\mid\; p = \langle v,v \rangle \,\wedge\, C(\langle v,v \rangle)\}} \mathcal{F}(p)]$

From [34] and [13], we have

(35) $\mathcal{S}^{k+1}_{\mathsf{pull+}}(v) = \mathcal{R}[$
$\mathcal{R}_{u \in \mathsf{preds}(v)}\ \mathcal{P}\left(\mathcal{R}_{p \,\in\, \{p \;\mid\; p \in \mathsf{Paths}(u) \,\wedge\, C(p) \,\wedge\, \mathsf{length}(p) < k\}} \mathcal{F}(p),\ \langle u, v \rangle\right),$
$\mathcal{R}_{p \,\in\, \{p \;\mid\; p = \langle v,v \rangle \,\wedge\, C(\langle v,v \rangle)\}} \mathcal{F}(p)]$

In the case that the size of the set of paths is more than one, from [8], and in the case that the set of paths is singleton, from $\mathcal{R}_{\{v\}} = v$ and [9], and in the case that the set of paths is empty, from $\mathcal{R}_{\emptyset} = \bot$ and [7],

(36) $\mathcal{P}\left(\mathcal{R}_{p \,\in\, \{p \;\mid\; p \in \mathsf{Paths}(u) \,\wedge\, C(p) \,\wedge\, \mathsf{length}(p) < k\}} \mathcal{F}(p),\ \langle u, v \rangle\right) =$

$$\mathcal{R}_{p \in \{p \mid p \in \mathsf{Paths}(u) \wedge C(p) \wedge \mathsf{length}(p) < k\}} \mathcal{F}(p \cdot \langle u, v \rangle))$$

From [35] and [36], we have

(37) $\mathcal{S}^{k+1}_{\mathsf{pull+}}(v) = \mathcal{R}[$

$$\mathcal{R}_{u \in \mathsf{preds}(v)}\ \mathcal{R}_{p \in \{p \mid p \in \mathsf{Paths}(u) \wedge C(p) \wedge \mathsf{length}(p) < k\}} \mathcal{F}(p \cdot \langle u, v \rangle),$$
$$\mathcal{R}_{p \in \{p \mid p = \langle v, v \rangle \wedge C(\langle v, v \rangle)\}} \mathcal{F}(p)]$$

that is

(38) $\mathcal{S}^{k+1}_{\mathsf{pull+}}(v) = \mathcal{R}[$

$$\mathcal{R}_{u \in \mathsf{preds}(v)}\ \mathcal{R}_{p' \in \{p' \mid p' = p \cdot \langle u, v \rangle \wedge p \in \mathsf{Paths}(u) \wedge C(p) \wedge \mathsf{length}(p) < k\}} \mathcal{F}(p'),$$
$$\mathcal{R}_{p \in \{p \mid p = \langle v, v \rangle \wedge C(\langle v, v \rangle)\}} \mathcal{F}(p)]$$

From [25] and Lemma 9,

(39) $\mathcal{S}^{k+1}_{\mathsf{pull+}}(v) = \mathcal{R}[$

$$\mathcal{R}_{u \in \mathsf{preds}(v)}\ \mathcal{R}_{p' \in \{p' \mid p' = p \cdot \langle u, v \rangle \wedge p \in \mathsf{Paths}(u) \wedge C(p \cdot \langle u, v \rangle) \wedge \mathsf{length}(p) < k\}} \mathcal{F}(p'),$$
$$\mathcal{R}_{p \in \{p \mid p = \langle v, v \rangle \wedge C(\langle v, v \rangle)\}} \mathcal{F}(p)]$$

From [39], [2] and [3], we have

(40) $\mathcal{S}^{k+1}_{\mathsf{pull+}}(v) = \mathcal{R}[$

$$\mathcal{R}_{p' \in \{p' \mid \exists u.\ u \in \mathsf{preds}(v) \wedge p' = p \cdot \langle u, v \rangle \wedge p \in \mathsf{Paths}(u) \wedge C(p \cdot \langle u, v \rangle) \wedge \mathsf{length}(p) < k\}} \mathcal{F}(p'),$$
$$\mathcal{R}_{p \in \{p \mid p = \langle v, v \rangle \wedge C(\langle v, v \rangle)\}} \mathcal{F}(p)]$$

that is

(41) $\mathcal{S}^{k+1}_{\mathsf{pull+}}(v) = \mathcal{R}[$

$$\mathcal{R}_{p' \in \{p' \mid p \in \mathsf{Paths}(v) \wedge C(p') \wedge 0 < \mathsf{length}(p') < k+1\}} \mathcal{F}(p'),$$
$$\mathcal{R}_{p \in \{p \mid p = \langle v, v \rangle \wedge C(\langle v, v \rangle)\}} \mathcal{F}(p)]$$

that is

$$\mathcal{S}^{k+1}_{\mathsf{pull+}}(v) = \mathcal{R}_{p' \in \{p' \mid p \in \mathsf{Paths}(v) \wedge C(p') \wedge \mathsf{length}(p') < k+1\}} \mathcal{F}(p')$$

LEMMA 9.

$\forall p, v.\ \mathsf{let}\ u := \mathsf{tail}(p)\ \mathsf{in}\ C(p) \leftrightarrow C(p \cdot \langle u, v \rangle)$

Proof.

We consider the two cases:

Case:

(1) $C(p) = (head(p) = s)$

Straightforward by

$head(p) = s \leftrightarrow head(p \cdot \langle u, v \rangle) = s$

(2) $C(p) = \mathsf{True}$

Straightforward by

$\mathsf{True} \leftrightarrow \mathsf{True}$

4.4.2 *Pull, Non-idempotent*

Theorem 23 (Correctness of Pull (non-idempotent reduction)).
For all $\mathcal{R}, \mathcal{F}, \mathcal{I}, \mathcal{P}, k \geq 1$ *and* s,
let $C(p) = (\text{head}(p) = s)$, *and*
there is no cycle that contains s,
if the conditions $\mathbb{C}_1$ - $\mathbb{C}_8$ *hold,*
$\mathcal{S}^k_{\text{pull-}}(v) = \mathcal{S}pec^k(v)$

Proof.
We assume that
(1) $\forall n.\ \mathcal{R}(n, \bot) = n$
(2) $\forall n, n'.\ \mathcal{R}(n, n') = \mathcal{R}(n', n)$
(3) $\forall n, n', n''.\ \mathcal{R}(\mathcal{R}(n, n'), n'') = \mathcal{R}(n, \mathcal{R}(n', n''))$
(4) $\forall v \in \mathsf{V}.\ C(\langle v, v\rangle) \longrightarrow \mathcal{I}(v) = \mathcal{F}(\langle v, v\rangle)$
(5) $\forall v \in \mathsf{V}.\ \neg C(\langle v, v\rangle) \longrightarrow \mathcal{I}(v) = \bot$
(6) $\forall e.\ \mathcal{P}(\bot, e) = \bot$
(7) $\forall p_1,\ p_2 \in \mathsf{P},\ v \in \mathsf{V}.$
$\text{tail}(p_1) = \text{tail}(p_2) \longrightarrow$
$\text{let } u := \text{tail}(p_1) \text{ in}$
$\mathcal{P}\ [\mathcal{R}(\mathcal{F}(p_1), \mathcal{F}(p_2)),\ \langle u, v\rangle] =$
$\mathcal{R}\ [\mathcal{F}(p_1 \cdot \langle u, v\rangle),\ \mathcal{F}(p_2 \cdot \langle u, v\rangle)]$
(8) $\forall p, e.\ \mathcal{P}(\mathcal{F}(p),\ e) = \mathcal{F}(p \cdot e)$
(9) $C(p) = (\text{head}(p) = s)$
(10) There is no cycle that contains s.

Form Def. 6, we have
$\mathcal{S}pec^k(v) = \mathcal{R}_{p \in \{p \mid p \in \text{Paths}(v) \wedge C(p) \wedge \text{length}(p) < k\}}\ \mathcal{F}(p)$

Proof by induction on k:
Base Case:
$k = 1$
We should show that
$\mathcal{S}^1_{\text{pull-}}(v) = \mathcal{R}_{p \in \{p \mid p \in \text{Paths}(v) \wedge C(p) \wedge \text{length}(p) < 1\}} \mathcal{F}(p)$
that is
$\mathcal{S}^1_{\text{pull-}}(v) = \mathcal{R}_{p \in \{p \mid p = \langle v, v\rangle \wedge C(p)\}} \mathcal{F}(p)$
We consider two cases:
Case:
(11) $C(\langle v, v\rangle)$
We should show that
$\mathcal{S}^1_{\text{pull-}}(v) = \mathcal{R}_{\{\langle v, v\rangle\}} \mathcal{F}(p)$
that is
$\mathcal{S}^1_{\text{pull-}}(v) = \mathcal{F}(\langle v, v\rangle)$
that is straightforward from Def. 2, [4] and [11].
Case:
(12) $\neg C(\langle v, v\rangle)$
We should show that
$\mathcal{S}^1_{\text{pull-}}(v) = \underset{\emptyset}{\mathcal{R}}\ \mathcal{F}(p)$

that is

$\mathcal{S}^{1}_{\text{pull}-}(v) = \bot$

that is straightforward from Def. 2, [5] and [12].

Inductive Case:

(13) $k > 1$

The induction hypothesis is:

(14) $\mathcal{S}^{k'}_{\text{pull}-}(v) = \mathcal{R}_{p \,\in\, \{p \;\mid\; p \in \mathsf{Paths}(v) \,\wedge\, C(p) \,\wedge\, \mathsf{length}(p) < k'\}} \mathcal{F}(p)$ for all v and $k' \leq k$

We should show that

$\mathcal{S}^{k+1}_{\text{pull}-}(v) = \mathcal{R}_{p \,\in\, \{p \;\mid\; p \in \mathsf{Paths}(v) \,\wedge\, C(p) \,\wedge\, \mathsf{length}(p) < k+1\}} \mathcal{F}(p)$

We consider two cases:

Case:

(15) $v = s$

By Lemma 11 on [9] and [4], [5], and [10],

(16) $\mathcal{S}^{k+1}_{\text{pull}-}(s) = \mathcal{I}(s)$

By [4] and [9],

(17) $\mathcal{I}(s) = \mathcal{F}(\langle s, s \rangle)$

From [16] and [17],

(18) $\mathcal{S}^{k+1}_{\text{pull}-}(s) = \mathcal{F}(\langle s, s \rangle)$

From [9] and [10],

(19) $\mathcal{R}_{p \,\in\, \{p \;\mid\; p \in \mathsf{Paths}(s) \,\wedge\, C(p) \,\wedge\, \mathsf{length}(p) < k+1\}} \mathcal{F}(p) =$
$\mathcal{R}_{p \,\in\, \{p \;\mid\; p \in \mathsf{Paths} \,\wedge\, \mathsf{tail}(p) = s \,\wedge\, \mathsf{head}(p) = s \,\wedge\, \mathsf{length}(p) < k+1\}} \mathcal{F}(p) =$
$\mathcal{R}_{p \,\in\, \{\langle s, s \rangle\}} \mathcal{F}(p) =$
$\mathcal{F}(\langle s, s \rangle)$

From [18] and [19],

$\mathcal{S}^{k+1}_{\text{pull}-}(v) = \mathcal{R}_{p \,\in\, \{p \;\mid\; p \in \mathsf{Paths}(v) \,\wedge\, C(p) \,\wedge\, \mathsf{length}(p) < k+1\}} \mathcal{F}(p)$

Case:

(20) $v \neq s$

From Def. 2, we consider two sub-cases:

Sub-case:

(21) $\mathsf{CPreds}^{k}(v) \neq \emptyset$

(22) $\mathcal{S}^{k+1}_{\text{pull}-}(v) = \mathcal{R}_{u \in \mathsf{preds}(v)} \, \mathcal{P}\left(\mathcal{S}^{k}_{\text{pull}-}(u), \langle u, v \rangle\right)$

From [22] and [14]

(23) $\mathcal{S}^{k+1}_{\text{pull}-}(v) = \mathcal{R}_{u \in \mathsf{preds}(v)} \left[\mathcal{P}\left(\mathcal{R}_{p \,\in\, \{p \;\mid\; p \in \mathsf{Paths}(u) \,\wedge\, C(p) \,\wedge\, \mathsf{length}(p) < k\}} \mathcal{F}(p),\ \langle u, v \rangle\right)\right]$

In the case that the size of the set of paths is more than one, from [7], and in the case that the set of paths is singleton, from $\mathcal{R}_{\{v\}} = v$ and [8], and in the case that the set of paths is empty, from $\mathcal{R}_{\emptyset} = \bot$ and [6], we have

(24) $\mathcal{P}\left(\mathcal{R}_{p \,\in\, \{p \;\mid\; p \in \mathsf{Paths}(u) \,\wedge\, C(p) \,\wedge\, \mathsf{length}(p) < k\}} \mathcal{F}(p),\ \langle u, v \rangle\right) =$
$\mathcal{R}_{p \,\in\, \{p \;\mid\; p \in \mathsf{Paths}(u) \,\wedge\, C(p) \,\wedge\, \mathsf{length}(p) < k\}} \mathcal{F}(p \cdot \langle u, v \rangle)$

After substituting [24] in [23]

(25) $\mathcal{S}^{k+1}_{\text{pull}-}(v) = \mathcal{R}_{u \in \mathsf{preds}(v)} \left[\mathcal{R}_{p \,\in\, \{p \;\mid\; p \in \mathsf{Paths}(u) \,\wedge\, C(p) \,\wedge\, \mathsf{length}(p) < k\}} \mathcal{F}(p \cdot \langle u, v \rangle)\right]$

that is

(26) $\mathcal{S}^{k+1}_{\text{pull}-}(v) = \mathcal{R}_{p \,\in\, \{p \;\mid\; \exists u.\ p \in \mathsf{Paths}(u) \,\wedge\, u \in \mathsf{preds}(v) \,\wedge\, C(p) \,\wedge\, \mathsf{length}(p) < k\}} \mathcal{F}(p \cdot \langle u, v \rangle)$

From [26] and Lemma 9

(27) $\mathcal{S}^{k+1}_{\text{pull}-}(v) = \mathcal{R}_{p \in \{p \mid \exists u.\ p \in \text{Paths}(u) \wedge u \in \text{preds}(v) \wedge C(p \cdot \langle u,v\rangle) \wedge \text{length}(p) < k\}} \mathcal{F}(p \cdot \langle u, v\rangle)$

that is

(28) $\mathcal{S}^{k+1}_{\text{pull}-}(v) = \mathcal{R}_{p' \in \{p' \mid p' \in \text{Paths}(v) \wedge C(p') \wedge 0 < \text{length}(p') < k+1\}} \mathcal{F}(p')$

From [28] and [20]

$\mathcal{S}^{k+1}_{\text{push}+}(v) = \mathcal{R}_{p \in \{p \mid p \in \text{Paths}(v) \wedge C(p) \wedge \text{length}(p) < k+1\}} \mathcal{F}(p),$

Sub-case:

(29) $\text{CPreds}^{k}(v) = \emptyset$

(30) $\mathcal{S}^{k+1}_{\text{pull}-}(v) = \mathcal{S}^{k}_{\text{pull}-}(v)$

(31) $\text{CPreds}^{k}(v) = \left\{ u \mid u \in \text{preds}(v) \wedge \mathcal{S}^{k}_{\text{pull}-}(u) \neq \mathcal{S}^{k-1}_{\text{pull}-}(u) \right\}$

From [30] and [14],

(32) $\mathcal{S}^{k+1}_{\text{pull}-}(v) = \mathcal{R}_{p \in \{p \mid p \in \text{Paths}(v) \wedge C(p) \wedge \text{length}(p) < k\}} \mathcal{F}(p)$

From [32] and [20],

(33) $\mathcal{S}^{k+1}_{\text{pull}-}(v) = \mathcal{R}_{p \in \{p \mid \exists u.\ p \in \text{Paths}(u) \wedge u \in \text{preds}(v) \wedge C(p \cdot \langle u,v\rangle) \wedge 0 < \text{length}(p \cdot \langle u,v\rangle) < k\}} \mathcal{F}(p)$

From [33] and Lemma 9,

(34) $\mathcal{S}^{k+1}_{\text{pull}-}(v) = \mathcal{R}_{p \in \{p \mid \exists u.\ p \in \text{Paths}(u) \wedge u \in \text{preds}(v) \wedge C(p) \wedge 0 < \text{length}(p \cdot \langle u,v\rangle) < k\}} \mathcal{F}(p)$

From [34], [2] and [3]

(35) $\mathcal{S}^{k+1}_{\text{pull}-}(v) = \mathcal{R}_{u \in \text{preds}(v)}\ \mathcal{R}_{p' \in \{p' \mid \exists u.\ p' \in \text{Paths}(u) \wedge C(p') \wedge \text{length}(p') < k-1\}} \mathcal{F}(p' \cdot \langle u, v\rangle)$

In the case that the size of the set of paths is more than one, from [7], and
in the case that the set of paths is singleton, from $\mathcal{R}_{\{v\}} = v$ and [8], and
in the case that the set of paths is empty, from $\mathcal{R}_{\emptyset} = \bot$ and [6],

(36) $\mathcal{P}\left(\mathcal{R}_{p \in \{p \mid p \in \text{Paths}(u) \wedge C(p) \wedge \text{length}(p) < k-1\}} \mathcal{F}(p),\ \langle u, v\rangle\right) =$
$\mathcal{R}_{p \in \{p \mid p \in \text{Paths}(u) \wedge C(p) \wedge \text{length}(p) < k-1\}} \mathcal{F}(p \cdot \langle u, v\rangle)$

From [35] and [36], we have

(37) $\mathcal{S}^{k+1}_{\text{pull}-}(v) = \mathcal{R}_{u \in \text{preds}(v)}\ \mathcal{P}\left(\mathcal{R}_{p \in \{p \mid p \in \text{Paths}(u) \wedge C(p) \wedge \text{length}(p) < k-1\}} \mathcal{F}(p),\ \langle u, v\rangle\right)$

From [37] and [14], we have

(38) $\mathcal{S}^{k+1}_{\text{pull}-}(v) = \mathcal{R}_{u \in \text{preds}(v)}\ \mathcal{P}\left(\mathcal{S}^{k-1}_{\text{pull}-}(u),\ \langle u, v\rangle\right)$

From [29] and [31], we have

(39) For all $u \in \text{preds}(v)$: $\mathcal{S}^{k}_{\text{pull}-}(u) = \mathcal{S}^{k-1}_{\text{pull}-}(u)$

From [38] and [39], we have

(40) $\mathcal{S}^{k+1}_{\text{pull}-}(v) = \mathcal{R}_{u \in \text{preds}(v)}\ \mathcal{P}\left(\mathcal{S}^{k}_{\text{pull}-}(u),\ \langle u, v\rangle\right)$

From [40] and [14], we have

(41) $\mathcal{S}^{k+1}_{\text{pull}-}(v) = \mathcal{R}_{u \in \text{preds}(v)}\ \mathcal{P}\left(\mathcal{R}_{p \in \{p \mid p \in \text{Paths}(u) \wedge C(p) \wedge \text{length}(p) < k\}} \mathcal{F}(p),\ \langle u, v\rangle\right)$

In the case that the size of the set of paths is more than one, from [7], and
in the case that the set of paths is singleton, from $\mathcal{R}_{\{v\}} = v$ and [8], and
in the case that the set of paths is empty, from $\mathcal{R}_{\emptyset} = \bot$ and [6],

(42) $\mathcal{P}\left(\mathcal{R}_{p \in \{p \mid p \in \text{Paths}(u) \wedge C(p) \wedge \text{length}(p) < k\}} \mathcal{F}(p),\ \langle u, v\rangle\right) =$
$\mathcal{R}_{p \in \{p \mid p \in \text{Paths}(u) \wedge C(p) \wedge \text{length}(p) < k\}} \mathcal{F}(p \cdot \langle u, v\rangle)$

From [41] and [42], we have

(43) $\mathcal{S}^{k+1}_{\text{pull}-}(v) = \mathcal{R}_{u \in \text{preds}(v)}\ \mathcal{R}_{p \in \{p \mid p \in \text{Paths}(u) \wedge C(p) \wedge \text{length}(p) < k\}} \mathcal{F}(p \cdot \langle u, v\rangle)$

that is

(44) $\mathcal{S}^{k+1}_{\text{pull}-}(v) = \mathcal{R}_{u \in \text{preds}(v)}\ \mathcal{R}_{p' \in \{p' \mid p' = p \cdot \langle u,v\rangle \wedge p \in \text{Paths}(u) \wedge C(p) \wedge \text{length}(p) < k\}} \mathcal{F}(p')$

From [31] and Lemma 9,

(45) $\mathcal{S}^{k+1}_{\text{pull}-}(v) = \mathcal{R}_{u \in \text{preds}(v)}\ \mathcal{R}_{p' \in \{p' \mid p' = p \cdot \langle u,v\rangle \wedge p \in \text{Paths}(u) \wedge C(p \cdot \langle u,v\rangle) \wedge \text{length}(p) < k\}} \mathcal{F}(p')$

From [45], [2] and [3], we have

(46) $\mathcal{S}^{k+1}_{\text{pull}-}(v) = \mathcal{R}_{p' \in \{p' \mid \exists u.\ u \in \text{preds}(v) \wedge p' = p \cdot \langle u,v \rangle \wedge p \in \text{Paths}(u) \wedge C(p \cdot \langle u,v \rangle) \wedge \text{length}(p) < k\}} \mathcal{F}(p')$

that is

(47) $\mathcal{S}^{k+1}_{\text{pull}-}(v) = \mathcal{R}_{p' \in \{p' \mid p \in \text{Paths}(v) \wedge C(p') \wedge 0 < \text{length}(p') < k+1\}} \mathcal{F}(p')$

From [47] and [20], we have

$\mathcal{S}^{k+1}_{\text{pull}-}(v) = \mathcal{R}_{p' \in \{p' \mid p \in \text{Paths}(v) \wedge C(p') \wedge \text{length}(p') < k+1\}} \mathcal{F}(p')$

Lemma 10.

For all $\mathcal{R}, \mathcal{F}, C, \mathcal{I}, \mathcal{P}, k \geq 1$ *and* s, v, *where*
(1) $C(p) = (\text{head}(p) = s)$,
(2) $\forall v \in \mathsf{V}.\ C(\langle v,v \rangle) \longrightarrow \mathcal{I}(v) = \mathcal{F}(\langle v,v \rangle)$
(3) $\forall v \in \mathsf{V}.\ \neg C(\langle v,v \rangle) \longrightarrow \mathcal{I}(v) = \bot$
if $\mathcal{S}^{k}_{\text{pull}-}(v) \neq \mathcal{S}^{k-1}_{\text{pull}-}(v)$,
then v *is reachable from* s.

Proof.
Immediate from induction on k and case analysis on branches of Def. 2.
The base case is from [1], [2] and [3].

Lemma 11.

For all $\mathcal{R}, \mathcal{F}, C, \mathcal{I}, \mathcal{P}, k \geq 1$ *and* s, v, *where*
(1) $C(p) = (\text{head}(p) = s)$,
(2) $\forall v \in \mathsf{V}.\ C(\langle v,v \rangle) \longrightarrow \mathcal{I}(v) = \mathcal{F}(\langle v,v \rangle)$
(3) $\forall v \in \mathsf{V}.\ \neg C(\langle v,v \rangle) \longrightarrow \mathcal{I}(v) = \bot$
there is no cycle that contains s,
then $\mathcal{S}^{k}_{\text{pull}-}(s) = \mathcal{I}(s)$.

Proof.
Immediate from induction on k and case analysis on branches of Def. 2.
The second branch is refuted by Lemma 10 and the assumption of acyclicity for s.

4.4.3 Push, Idempotent

Theorem 24 (Correctness of Push (idempotent reduction)).
For all $\mathcal{R}, \mathcal{F}, C, \mathcal{I}, \mathcal{P}$, *and* $k \geq 1$, *if the conditions* $\mathbb{C}_1$ - $\mathbb{C}_9$ *hold,*
$\mathcal{S}^k_{\text{push+}}(v) = \mathcal{S}pec^k(v)$

Proof.
We assume that

(1) $\forall n.\, \mathcal{R}(n, \bot) = n$
(2) $\forall n, n'.\, \mathcal{R}(n, n') = \mathcal{R}(n', n)$
(3) $\forall n, n', n''.\, \mathcal{R}(\mathcal{R}(n, n'), n'') = \mathcal{R}(n, \mathcal{R}(n', n''))$
(4) $\forall n.\, \mathcal{R}(n, n) = n$
(5) $\forall v \in \mathsf{V}.\, C(\langle v, v \rangle) \longrightarrow \mathcal{I}(v) = \mathcal{F}(\langle v, v \rangle)$
(6) $\forall v \in \mathsf{V}.\, \neg C(\langle v, v \rangle) \longrightarrow \mathcal{I}(v) = \bot$
(7) $\forall e.\, \mathcal{P}(\bot, e) = \bot$
(8) $\forall p_1,\ p_2 \in \mathsf{P},\ v \in \mathsf{V}.$
$\quad \mathsf{tail}(p_1) = \mathsf{tail}(p_2) \longrightarrow$
$\quad \mathsf{let}\ u := \mathsf{tail}(p_1)\ \mathsf{in}$
$\quad \mathcal{P}\,[\mathcal{R}(\mathcal{F}(p_1), \mathcal{F}(p_2)),\ \langle u, v \rangle] =$
$\quad \mathcal{R}\,[\mathcal{F}(p_1 \cdot \langle u, v \rangle),\ \mathcal{F}(p_2 \cdot \langle u, v \rangle)]$
(9) $\forall p, e.\, \mathcal{P}(\mathcal{F}(p),\ e) = \mathcal{F}(p \cdot e)$

Form Def. 6, we have
$\mathcal{S}pec^k(v) = \mathcal{R}_{p \,\in\, \{p \,|\, p \in \mathsf{Paths}(v)\ \wedge\ C(p)\ \wedge\ \mathsf{length}(p) < k\}}\ \mathcal{F}(p)$

Proof by induction on k:
Base Case:
We should show that
$\mathcal{S}^1_{\text{push+}}(v) = \mathcal{R}_{p \,\in\, \{p \ |\ p \in \mathsf{Paths}(v)\ \wedge\ C(p)\ \wedge\ \mathsf{length}(p) < 1\}} \mathcal{F}(p)$
that is
$\mathcal{S}^1_{\text{push+}}(v) = \mathcal{R}_{p \,\in\, \{p \ |\ p = \langle v, v \rangle\ \wedge\ C(p)\}} \mathcal{F}(p)$
We consider two cases:
Case:
(10) $C(\langle v, v \rangle)$
We should show that
$\mathcal{S}^1_{\text{push+}}(v) = \mathcal{R}_{\{\langle v, v \rangle\}} \mathcal{F}(p)$
that is
$\mathcal{S}^1_{\text{push+}}(v) = \mathcal{F}(\langle v, v \rangle)$
that is straightforward from Def. 3, [5] and [10].
Case:
(11) $\neg C(\langle v, v \rangle)$
We should show that
$\mathcal{S}^1_{\text{push+}}(v) = \underset{\emptyset}{\mathcal{R}}\, \mathcal{F}(p)$
that is
$\mathcal{S}^1_{\text{push+}}(v) = \bot$
that is straightforward from Def. 3, [6] and [11].

Inductive Case:

The induction hypothesis is:

(12) $\mathcal{S}^{k}_{\text{push+}}(v) = \mathcal{R}_{p \in \{p \mid p \in \mathsf{Paths}(v) \wedge C(p) \wedge \mathsf{length}(p) < k\}} \mathcal{F}(p), \quad k > 1$

We should show that

$\mathcal{S}^{k+1}_{\text{push+}}(v) = \mathcal{R}_{p \in \{p \mid p \in \mathsf{Paths}(v) \wedge C(p) \wedge \mathsf{length}(p) < k+1\}} \mathcal{F}(p)$

From Def. 3, we have that

(13) $\mathcal{S}^{k+1}_{\text{push+}}(v) = S_n$

(14) $\{u_1, .., u_n\} = u \in \left\{u \mid u \in \mathsf{preds}(v) \;\wedge\; \mathcal{S}^{k}_{\text{push+}}(u) \neq \mathcal{S}^{k-1}_{\text{push+}}(u)\right\}$

(15) $S_0 = \mathcal{S}^{k}_{\text{push+}}(v)$

(16) $S_{i+1} = \mathcal{R}\left(S_i,\ \mathcal{P}(\mathcal{S}^{k}_{\text{push+}}(u_i), \langle u_i, v\rangle)\right)$

From [13]-[16], and [2] and [3], we have

(17) $\mathcal{S}^{k+1}_{\text{push+}}(v) = \mathcal{R}\left[\mathcal{R}_{u \in \{u \mid u \in \mathsf{preds}(v) \wedge \mathcal{S}^{k}_{\text{push+}}(u) \neq \mathcal{S}^{k-1}_{\text{push+}}(u)\}} \left[\mathcal{P}(\mathcal{S}^{k}_{\text{push+}}(u), \langle u, v\rangle)\right],\ \mathcal{S}^{k}_{\text{push+}}(v)\right]$

From Lemma 12, and [2], [3], and [4], we have

(18) $\mathcal{S}^{k}_{\text{push+}}(v) = \mathcal{R}\left(\mathcal{S}^{k}_{\text{push+}}(v), \mathcal{R}_{u \in \{u \mid u \in \mathsf{preds}(v) \wedge \mathcal{S}^{k}_{\text{push+}}(u) = \mathcal{S}^{k-1}_{\text{push+}}(u)\}} \mathcal{P}(\mathcal{S}^{k}_{\text{push+}}(u), \langle u, v\rangle)\right)$

After substituting [18] in [17]

(19) $\mathcal{S}^{k+1}_{\text{push+}}(v) = \mathcal{R}[\mathcal{R}_{u \in \{u \mid u \in \mathsf{preds}(v) \wedge \mathcal{S}^{k}_{\text{push+}}(u) \neq \mathcal{S}^{k-1}_{\text{push+}}(u)\}} \left[\mathcal{P}(\mathcal{S}^{k}_{\text{push+}}(u), \langle u, v\rangle)\right],$
$\mathcal{R}(\mathcal{S}^{k}_{\text{push+}}(v), \mathcal{R}_{u \in \{u \mid u \in \mathsf{preds}(v) \wedge \mathcal{S}^{k}_{\text{push+}}(u) = \mathcal{S}^{k-1}_{\text{push+}}(u)\}} \left[\mathcal{P}(\mathcal{S}^{k}_{\text{push+}}(u), \langle u, v\rangle)\right])]$

From [19], [2] and [3]

(20) $\mathcal{S}^{k+1}_{\text{push+}}(v) = \mathcal{R}\left[\mathcal{R}_{u \in \mathsf{preds}(v)} \left[\mathcal{P}(\mathcal{S}^{k}_{\text{push+}}(u), \langle u, v\rangle)\right],\ \mathcal{S}^{k}_{\text{push+}}(v)\right]$

From [20] and [12]

(21) $\mathcal{S}^{k+1}_{\text{push+}}(v) = \mathcal{R}[$
$\mathcal{R}_{u \in \mathsf{preds}(v)} \left[\mathcal{P}\left(\mathcal{R}_{p \in \{p \mid p \in \mathsf{Paths}(u) \wedge C(p) \wedge \mathsf{length}(p) < k\}} \mathcal{F}(p), \langle u, v\rangle\right)\right],$
$\mathcal{S}^{k}_{\text{push+}}(v)]$

In the case that the size of the set of paths is more than one, from [8], and in the case that the set of paths is singleton, from $\mathcal{R}_{\{v\}} = v$ and [9], and in the case that the set of paths is empty, from $\mathcal{R}_{\emptyset} = \bot$ and [7], we have

(22) $\mathcal{P}(\mathcal{R}_{p \in \{p \mid p \in \mathsf{Paths}(u) \wedge C(p) \wedge \mathsf{length}(p) < k\}} \mathcal{F}(p), \langle u, v\rangle) =$
$\mathcal{R}_{p \in \{p \mid p \in \mathsf{Paths}(u) \wedge C(p) \wedge \mathsf{length}(p) < k\}} \mathcal{F}(p \cdot \langle u, v\rangle)$

After substituting [22] in [21]

(23) $\mathcal{S}^{k+1}_{\text{push+}}(v) = \mathcal{R}[$
$\mathcal{R}_{u \in \mathsf{preds}(v)} \left[\mathcal{R}_{p \in \{p \mid p \in \mathsf{Paths}(u) \wedge C(p) \wedge \mathsf{length}(p) < k\}} \mathcal{F}(p \cdot \langle u, v\rangle)\right],$
$\mathcal{S}^{k}_{\text{push+}}(v)]$

that is

(24) $\mathcal{S}^{k+1}_{\text{push+}}(v) = \mathcal{R}[$
$\mathcal{R}_{p \in \{p \mid \exists u.\ p \in \mathsf{Paths}(u) \wedge u \in \mathsf{preds}(v) \wedge C(p) \wedge \mathsf{length}(p) < k\}} \mathcal{F}(p \cdot \langle u, v\rangle),$
$\mathcal{S}^{k}_{\text{push+}}(v)]$

From [24] and Lemma 9

(25) $\mathcal{S}^{k+1}_{\text{push+}}(v) = \mathcal{R}[$
$\mathcal{R}_{p \in \{p \mid \exists u.\ p \in \mathsf{Paths}(u) \wedge u \in \mathsf{preds}(v) \wedge C(p \cdot \langle u, v\rangle) \wedge \mathsf{length}(p) < k\}} \mathcal{F}(p \cdot \langle u, v\rangle),$
$\mathcal{S}^{k}_{\text{push+}}(v)]$

that is

(26) $\mathcal{S}^{k+1}_{\text{push+}}(v) = \mathcal{R}[$

$\mathcal{R}_{p' \in \{p' \mid p' \in \mathsf{Paths}(v) \wedge C(p') \wedge \mathsf{length}(p') < k+1\}} \mathcal{F}(p'),$
$\mathcal{S}^{k}_{\text{push+}}(v)]$

From [26] and [12]

(27) $\mathcal{S}^{k+1}_{\text{push+}}(v) = \mathcal{R}[$
$\mathcal{R}_{p \in \{p \mid p \in \mathsf{Paths}(v) \wedge C(p) \wedge \mathsf{length}(p) < k+1\}} \mathcal{F}(p),$
$\mathcal{R}_{p \in \{p \mid p \in \mathsf{Paths}(v) \wedge C(p) \wedge \mathsf{length}(p) < k\}} \mathcal{F}(p)]$

From [27] and [4]

(28) $\mathcal{S}^{k+1}_{\text{push+}}(v) = \mathcal{R}_{p \in \{p \mid p \in \mathsf{Paths}(v) \wedge C(p) \wedge \mathsf{length}(p) < k+1\}} \mathcal{F}(p),$

Lemma 12.

For all $\mathcal{R}, \mathcal{F}, C, \mathcal{I}, \mathcal{P}$, *if the conditions* $\mathbb{C}_1$ - $\mathbb{C}_{10}$ *hold,*
$\forall v, u, k.$
$k \geq 1 \;\wedge\; u \in \mathsf{preds}(v) \;\wedge\; \mathcal{S}^{k}_{\text{push+}}(u) = \mathcal{S}^{k-1}_{\text{push+}}(u) \rightarrow$
$\mathcal{S}^{k}_{\text{push+}}(v) = \mathcal{R}(\mathcal{S}^{k}_{\text{push+}}(v), \mathcal{P}(\mathcal{S}^{k}_{\text{push+}}(u), \langle u, v\rangle))$

Proof.

We assume that

(1) $\forall n.\, \mathcal{R}(n, \bot) = n$
(2) $\forall n, n'.\, \mathcal{R}(n, n') = \mathcal{R}(n', n)$
(3) $\forall n, n', n''.\, \mathcal{R}(\mathcal{R}(n, n'), n'') = \mathcal{R}(n, \mathcal{R}(n', n''))$
(4) $\forall n.\, \mathcal{R}(n, n) = n$
(5) $\forall v \in \mathsf{V}.\, C(\langle v, v\rangle) \longrightarrow \mathcal{I}(v) = \mathcal{F}(\langle v, v\rangle)$
(6) $\forall v \in \mathsf{V}.\, \neg C(\langle v, v\rangle) \longrightarrow \mathcal{I}(v) = \bot$
(7) $\forall e.\, \mathcal{P}(\bot, e) = \bot$
(8) $\forall p_1,\ p_2 \in \mathsf{P},\ v \in \mathsf{V}.$
$C(p_1) \wedge C(p_2) \wedge$
$\mathsf{tail}(p_1) = \mathsf{tail}(p_2) \longrightarrow$
$\mathsf{let}\ u := \mathsf{tail}(p_1)\ \mathsf{in}$
$\mathcal{P}\,[\mathcal{R}(\mathcal{F}(p_1), \mathcal{F}(p_2)),\ \langle u, v\rangle] =$
$\mathcal{R}\,[\mathcal{F}(p_1 \cdot \langle u, v\rangle),\ \mathcal{F}(p_2 \cdot \langle u, v\rangle)]$
(9) $\forall p, e.\, \mathcal{P}(\mathcal{F}(p),\ e) = \mathcal{F}(p \cdot e)$

Proof by induction on k:

Base Case:

(10) $k = 1$

We assume that

(11) $u \in \mathsf{preds}(v)$
(12) $\mathcal{S}^{1}_{\text{push+}}(u) = \mathcal{S}^{0}_{\text{push+}}(u)$

From Def. 3 on [12]

(13) $\mathcal{S}^{1}_{\text{push+}}(u) = \bot$

We need to show that

(14) $\mathcal{S}^{1}_{\text{push+}}(v) = \mathcal{R}(\mathcal{S}^{1}_{\text{push+}}(v), \mathcal{P}(\mathcal{S}^{1}_{\text{push+}}(u), \langle u, v\rangle))$

From [13] and [7], we need to show that

(15) $\mathcal{S}^{1}_{\text{push+}}(v) = \mathcal{R}(\mathcal{S}^{1}_{\text{push+}}(v), \bot)$

that is immediate from [1].

Inductive Case:

The induction hypothesis is

(16) $\forall v, u.$

$$u \in \mathsf{preds}(v) \ \wedge \ \mathcal{S}^{k-1}_{\text{push+}}(u) = \mathcal{S}^{k-2}_{\text{push+}}(u) \rightarrow$$
$$\mathcal{S}^{k-1}_{\text{push+}}(v) = \mathcal{R}(\mathcal{S}^{k-1}_{\text{push+}}(v), \mathcal{P}(\mathcal{S}^{k-1}_{\text{push+}}(u), \langle u, v\rangle))$$

We assume that

(17) $k > 1$

(18) $u \in \mathsf{preds}(v)$

(19) $\mathcal{S}^{k}_{\text{push+}}(u) = \mathcal{S}^{k-1}_{\text{push+}}(u)$

We show that

$$\mathcal{S}^{k}_{\text{push+}}(v) = \mathcal{R}(\mathcal{S}^{k}_{\text{push+}}(v), \mathcal{P}(\mathcal{S}^{k}_{\text{push+}}(u), \langle u, v\rangle))$$

From Def. 3 on [17], we have that

(20) $\mathcal{S}^{k}_{\text{push+}}(v) = S_n$

(21) $\{u_1, .., u_n\} = u \ \in \ \left\{u \mid u \in \mathsf{preds}(v) \ \wedge \ \mathcal{S}^{k-1}_{\text{push+}}(u) \neq \mathcal{S}^{k-2}_{\text{push+}}(u)\right\}$

(22) $S_0 = \mathcal{S}^{k-1}_{\text{push+}}(v)$

(23) $S_{i+1} = \mathcal{R}\left(S_i, \ \mathcal{P}(\mathcal{S}^{k-1}_{\text{push+}}(u_i), \langle u_i, v\rangle)\right)$

From [20]-[23], and [2] and [3], we have

(24) $\mathcal{S}^{k}_{\text{push+}}(v) = \mathcal{R}\left[\mathcal{R}_{u \,\in\, \{u \ \mid \ u \in \mathsf{preds}(v) \ \wedge \ \mathcal{S}^{k-2}_{\text{push+}}(u) \neq \mathcal{S}^{k-1}_{\text{push+}}(u)\}} \left[\mathcal{P}(\mathcal{S}^{k-1}_{\text{push+}}(u), \langle u, v\rangle)\right], \ \mathcal{S}^{k-1}_{\text{push+}}(v)\right]$

We consider two cases:

Case:

(25) $\mathcal{S}^{k-2}_{\text{push+}}(u) \neq \mathcal{S}^{k-1}_{\text{push+}}(u)$

From [24], [25], and [4] we have

(26) $\mathcal{S}^{k}_{\text{push+}}(v) = \mathcal{R}\left(\mathcal{S}^{k}_{\text{push+}}(v), \mathcal{P}(\mathcal{S}^{k-1}_{\text{push+}}(u), \langle u, v\rangle)\right)$

From [26] and [19]

$$\mathcal{S}^{k}_{\text{push+}}(v) = \mathcal{R}\left(\mathcal{S}^{k}_{\text{push+}}(v), \mathcal{P}(\mathcal{S}^{k}_{\text{push+}}(u), \langle u, v\rangle)\right)$$

Case:

(27) $\mathcal{S}^{k-2}_{\text{push+}}(u) = \mathcal{S}^{k-1}_{\text{push+}}(u)$

From [24] and [4] we have

(28) $\mathcal{S}^{k}_{\text{push+}}(v) = \mathcal{R}\left(\mathcal{S}^{k}_{\text{push+}}(v), \mathcal{S}^{k-1}_{\text{push+}}(v)\right)$

From [27] and [19], we have

(29) $\mathcal{S}^{k-1}_{\text{push+}}(u) = \mathcal{S}^{k-2}_{\text{push+}}(u)$

From [16] on [18] and [29] and then [19], we have

(30) $\mathcal{S}^{k-1}_{\text{push+}}(v) = \mathcal{R}(\mathcal{S}^{k-1}_{\text{push+}}(v), \mathcal{P}(\mathcal{S}^{k}_{\text{push+}}(u), \langle u, v\rangle))$

From [28], [30] and [2], we have

$$\mathcal{S}^{k}_{\text{push+}}(v) = \mathcal{R}(\mathcal{S}^{k}_{\text{push+}}(v), \mathcal{P}(\mathcal{S}^{k}_{\text{push+}}(u), \langle u, v\rangle))$$

4.4.4 Push, Non-idempotent

We consider the two variants in turn.
The first variant of push, non-idempotent was defined in Fig. 8, Def. 4.

Theorem 25 (Correctness of Push (non-idempotent reduction) I).
For all $\mathcal{R}, \mathcal{F}, \mathcal{I}, \mathcal{P}, k \geq 1$, *and* s,
let $C(p) = (\mathsf{head}(p) = s)$,
if the conditions $\mathbb{C}_1$ - $\mathbb{C}_8$ *hold, and*
s *is not on any cycle,*
$\mathcal{S}^k_{\mathsf{push-}}(v) = \mathcal{S}pec^k(v)$

Proof.
We assume that

(1) $\forall n.\ \mathcal{R}(n, \bot) = n$
(2) $\forall n, n'.\ \mathcal{R}(n, n') = \mathcal{R}(n', n)$
(3) $\forall n, n', n''.\ \mathcal{R}(\mathcal{R}(n, n'), n'') = \mathcal{R}(n, \mathcal{R}(n', n''))$
(4) $\forall v \in \mathsf{V}.\ C(\langle v, v\rangle) \longrightarrow \mathcal{I}(v) = \mathcal{F}(\langle v, v\rangle)$
(5) $\forall v \in \mathsf{V}.\ \neg C(\langle v, v\rangle) \longrightarrow \mathcal{I}(v) = \bot$
(6) $\forall e.\ \mathcal{P}(\bot, e) = \bot$
(7) $\forall p_1,\ p_2 \in \mathsf{P},\ v \in \mathsf{V}.$
$\quad \mathsf{tail}(p_1) = \mathsf{tail}(p_2) \longrightarrow$
$\quad \mathsf{let}\ u := \mathsf{tail}(p_1)\ \mathsf{in}$
$\quad \mathcal{P}\ [\mathcal{R}(\mathcal{F}(p_1), \mathcal{F}(p_2)),\ \langle u, v\rangle] =$
$\quad \mathcal{R}\ [\mathcal{F}(p_1 \cdot \langle u, v\rangle),\ \mathcal{F}(p_2 \cdot \langle u, v\rangle)]$
(8) $\forall p, e.\ \mathcal{P}(\mathcal{F}(p),\ e) = \mathcal{F}(p \cdot e)$
(9) $C(p) = (\mathsf{head}(p) = s)$
(10) There is no cycle that contains s.

Form Def. 6, we have
$$\mathcal{S}pec^k(v) = \mathcal{R}_{p\, \in\, \{p \mid p \in \mathsf{Paths}(v)\ \wedge\ C(p)\ \wedge\ \mathsf{length}(p) < k\}}\ \mathcal{F}(p)$$

Proof by induction on k:
Base Case:
$k = 1$
We should show that
$$\mathcal{S}^1_{\mathsf{push-}}(v) = \mathcal{R}_{p\, \in\, \{p\ \mid\ p \in \mathsf{Paths}(v)\ \wedge\ C(p)\ \wedge\ \mathsf{length}(p) < 1\}} \mathcal{F}(p)$$
that is
$$\mathcal{S}^1_{\mathsf{push-}}(v) = \mathcal{R}_{p\, \in\, \{p\ \mid\ p = \langle v, v\rangle\ \wedge\ C(p)\}} \mathcal{F}(p)$$
We consider two cases:
Case:
(11) $C(\langle v, v\rangle)$
We should show that
$$\mathcal{S}^1_{\mathsf{push-}}(v) = \mathcal{R}_{\{\langle v, v\rangle\}} \mathcal{F}(p)$$
that is
$$\mathcal{S}^1_{\mathsf{push-}}(v) = \mathcal{F}(\langle v, v\rangle)$$
that is straightforward from Def. 4, [4] and [11].
Case:
(12) $\neg C(\langle v, v\rangle)$

We should show that

$\mathcal{S}^{1}_{\text{push}-}(v) = \underset{\emptyset}{\mathcal{R}}\,\mathcal{F}(p)$

that is

$\mathcal{S}^{1}_{\text{push}-}(v) = \bot$

that is straightforward from Def. 4, [5] and [12].

Inductive Case:

(13) $k > 1$

The induction hypothesis is:

(14) $\mathcal{S}^{k'}_{\text{push}-}(v) = \mathcal{R}_{p \,\in\, \{p \;\mid\; p \in \text{Paths}(v) \,\wedge\, C(p) \,\wedge\, \text{length}(p) < k'\}} \mathcal{F}(p)$ for all v and $k' \leq k$

We should show that

$\mathcal{S}^{k+1}_{\text{push}-}(v) = \mathcal{R}_{p \,\in\, \{p \;\mid\; p \in \text{Paths}(v) \,\wedge\, C(p) \,\wedge\, \text{length}(p) < k+1\}} \mathcal{F}(p)$

We consider two cases:

Case:

(15) $v = s$

By Lemma 14 on [9] and [4], [5] and [10],

(16) $\mathcal{S}^{k+1}_{\text{push}-}(s) = \mathcal{I}(s)$

By [4] and [9],

(17) $\mathcal{I}(s) = \mathcal{F}(\langle s, s \rangle)$

From [16] and [17],

(18) $\mathcal{S}^{k+1}_{\text{push}-}(s) = \mathcal{F}(\langle s, s \rangle)$

From [9] and [10],

(19) $\mathcal{R}_{p \,\in\, \{p \;\mid\; p \in \text{Paths}(s) \,\wedge\, C(p) \,\wedge\, \text{length}(p) < k+1\}} \mathcal{F}(p) =$
$\mathcal{R}_{p \,\in\, \{p \;\mid\; p \in \text{Paths} \,\wedge\, \text{tail}(p) = s \,\wedge\, \text{head}(p) = s \,\wedge\, \text{length}(p) < k+1\}} \mathcal{F}(p) =$
$\mathcal{R}_{p \,\in\, \{\langle s, s \rangle\}} \mathcal{F}(p) =$
$\mathcal{F}(\langle s, s \rangle)$

From [18] and [19],

$\mathcal{S}^{k+1}_{\text{push}-}(v) = \mathcal{R}_{p \,\in\, \{p \;\mid\; p \in \text{Paths}(v) \,\wedge\, C(p) \,\wedge\, \text{length}(p) < k+1\}} \mathcal{F}(p)$

Case:

(20) $v \neq s$

From Def. 4, [1], [2], and [3], we have

(21) $\mathcal{S}^{k+1}_{\text{push}-}(v) = \mathcal{R}_{u \in \text{preds}(v)}\, \mathcal{P}\left(\mathcal{S}^{k}_{\text{push}-}(u), \langle u, v \rangle\right)$

From [21] and [14]

(22) $\mathcal{S}^{k+1}_{\text{push}-}(v) = \mathcal{R}_{u \in \text{preds}(v)} \left[\mathcal{P}\left(\mathcal{R}_{p \,\in\, \{p \;\mid\; p \in \text{Paths}(u) \,\wedge\, C(p) \,\wedge\, \text{length}(p) < k\}} \mathcal{F}(p),\ \langle u, v \rangle\right)\right]$

In the case that the size of the set of paths is more than one, from [7], and in the case that the set of paths is singleton, from $\mathcal{R}_{\{v\}} = v$ and [8], and in the case that the set of paths is empty, from $\mathcal{R}_{\emptyset} = \bot$ and [6], we have

(23) $\mathcal{P}\left(\mathcal{R}_{p \,\in\, \{p \;\mid\; p \in \text{Paths}(u) \,\wedge\, C(p) \,\wedge\, \text{length}(p) < k\}} \mathcal{F}(p),\ \langle u, v \rangle\right) =$
$\mathcal{R}_{p \,\in\, \{p \;\mid\; p \in \text{Paths}(u) \,\wedge\, C(p) \,\wedge\, \text{length}(p) < k\}} \mathcal{F}(p \cdot \langle u, v \rangle)$

After substituting [23] in [22]

(24) $\mathcal{S}^{k+1}_{\text{push}-}(v) = \mathcal{R}_{u \in \text{preds}(v)} \left[\mathcal{R}_{p \,\in\, \{p \;\mid\; p \in \text{Paths}(u) \,\wedge\, C(p) \,\wedge\, \text{length}(p) < k\}} \mathcal{F}(p \cdot \langle u, v \rangle)\right]$

that is

(25) $\mathcal{S}^{k+1}_{\text{push}-}(v) = \mathcal{R}_{p \,\in\, \{p \;\mid\; \exists u.\ p \in \text{Paths}(u) \,\wedge\, u \in \text{preds}(v) \,\wedge\, C(p) \,\wedge\, \text{length}(p) < k\}} \mathcal{F}(p \cdot \langle u, v \rangle)$

From [25] and Lemma 9

(26) $\mathcal{S}^{k+1}_{\text{push}-}(v) = \mathcal{R}_{p \in \{p \mid \exists u.\ p \in \text{Paths}(u) \land u \in \text{preds}(v) \land C(p \cdot \langle u,v \rangle) \land \text{length}(p) < k\}} \mathcal{F}(p \cdot \langle u, v \rangle)$

that is

(27) $\mathcal{S}^{k+1}_{\text{push}-}(v) = \mathcal{R}_{p' \in \{p' \mid p' \in \text{Paths}(v) \land C(p') \land 0 < \text{length}(p') < k+1\}} \mathcal{F}(p')$

From [27] and [20]

$\mathcal{S}^{k+1}_{\text{push}+}(v) = \mathcal{R}_{p \in \{p \mid p \in \text{Paths}(v) \land C(p) \land \text{length}(p) < k+1\}} \mathcal{F}(p)$,

LEMMA 13.

For all $\mathcal{R}, \mathcal{F}, C, \mathcal{I}, \mathcal{P}, k \geq 1$ *and* s, v, *where*

(1) $C(p) = (\text{head}(p) = s)$,

(2) $\forall v \in \text{V}.\ C(\langle v, v \rangle) \longrightarrow \mathcal{I}(v) = \mathcal{F}(\langle v, v \rangle)$

(3) $\forall v \in \text{V}.\ \neg C(\langle v, v \rangle) \longrightarrow \mathcal{I}(v) = \bot$

if $\mathcal{S}^{k}_{\text{push}-}(v) \neq \mathcal{S}^{k-1}_{\text{push}-}(v)$,

then v *is reachable from* s.

Proof.

Immediate from induction on k for Def. 4.

The base case is from [1], [2] and [3].

LEMMA 14.

For all $\mathcal{R}, \mathcal{F}, C, \mathcal{I}, \mathcal{P}, k \geq 1$ *and* s, v, *where*

(1) $C(p) = (\text{head}(p) = s)$,

(2) $\forall v \in \text{V}.\ C(\langle v, v \rangle) \longrightarrow \mathcal{I}(v) = \mathcal{F}(\langle v, v \rangle)$

(3) $\forall v \in \text{V}.\ \neg C(\langle v, v \rangle) \longrightarrow \mathcal{I}(v) = \bot$

there is no cycle that contains s,

then $\mathcal{S}^{k}_{\text{push}-}(s) = \mathcal{I}(s)$.

Proof.

Immediate from induction on k for Def. 4.

The inductive case is refuted by Lemma 13 and the assumption of acyclicity for s.

We now consider the second variant. The second variant of push, non-idempotent was defined in Def. 7.

Theorem 26 (Correctness of Push (non-idempotent reduction) II).
For all $\mathcal{R}$, $\mathcal{F}$, C, $\mathcal{I}$, $\mathcal{P}$, *and* $k \geq 1$, *if the conditions* $\mathbb{C}_1$ - $\mathbb{C}_8$ *and* $\mathbb{C}_{11}$ *hold,*
$\mathcal{S}^k_{\text{push}-}(v) = \mathcal{S}pec^k(v)$

We assume that

(1) $\forall n.\ \mathcal{R}(n, \bot) = n$
(2) $\forall n, n'.\ \mathcal{R}(n, n') = \mathcal{R}(n', n)$
(3) $\forall n, n', n''.\ \mathcal{R}(\mathcal{R}(n, n'), n'') = \mathcal{R}(n, \mathcal{R}(n', n''))$
(4) $\forall v \in \mathsf{V}.\ C(\langle v, v\rangle) \longrightarrow \mathcal{I}(v) = \mathcal{F}(\langle v, v\rangle)$
(5) $\forall v \in \mathsf{V}.\ \neg C(\langle v, v\rangle) \longrightarrow \mathcal{I}(v) = \bot$
(6) $\forall e.\ \mathcal{P}(\bot, e) = \bot$
(7) $\forall p_1,\ p_2 \in \mathsf{P},\ v \in \mathsf{V}.$
$\quad \mathsf{tail}(p_1) = \mathsf{tail}(p_2) \longrightarrow$
$\quad \mathsf{let}\ u := \mathsf{tail}(p_1)\ \mathsf{in}$
$\quad \mathcal{P}\,[\mathcal{R}(\mathcal{F}(p_1), \mathcal{F}(p_2)),\ \langle u, v\rangle] =$
$\quad \mathcal{R}\,[\mathcal{F}(p_1 \cdot \langle u, v\rangle),\ \mathcal{F}(p_2 \cdot \langle u, v\rangle)]$
(8) $\forall p, e.\ \mathcal{P}(\mathcal{F}(p),\ e) = \mathcal{F}(p \cdot e)$
(9) $\forall n, n'.\ \mathcal{R}(n, \mathcal{R}(\mathcal{P}(n', \langle u, v\rangle),$
$\qquad \mathcal{B}\,(n', \langle u, v\rangle))) = n$

Form Def. 6, we have

$$\mathcal{S}pec^k(v) = \mathcal{R}_{p\,\in\,\{p\ |\ p \in \mathsf{Paths}(v)\ \wedge\ C(p)\ \wedge\ \mathsf{length}(p) < k\}}\ \mathcal{F}(p)$$

Proof by induction on k:

Base Case:

$k = 1$

We should show that

$$\mathcal{S}^1_{\text{push}-}(v) = \mathcal{R}_{p\,\in\,\{p\ |\ p \in \mathsf{Paths}(v)\ \wedge\ C(p)\ \wedge\ \mathsf{length}(p) < 1\}} \mathcal{F}(p)$$

that is

$$\mathcal{S}^1_{\text{push}-}(v) = \mathcal{R}_{p\,\in\,\{p\ |\ p = \langle v, v\rangle\ \wedge\ C(p)\}} \mathcal{F}(p)$$

We consider two cases:

Case:

(10) $C(\langle v, v\rangle)$

We should show that

$$\mathcal{S}^1_{\text{push}-}(v) = \mathcal{R}_{\{\langle v, v\rangle\}} \mathcal{F}(p)$$

that is

$$\mathcal{S}^1_{\text{push}-}(v) = \mathcal{F}(\langle v, v\rangle)$$

that is straightforward from Def. 4, [4] and [10].

Case:

(11) $\neg C(\langle v, v\rangle)$

We should show that

$$\mathcal{S}^1_{\text{push}-}(v) = \underset{\emptyset}{\mathcal{R}}\ \mathcal{F}(p)$$

that is

$$\mathcal{S}^1_{\text{push}-}(v) = \bot$$

that is straightforward from Def. 4, [5] and [11].

Inductive Case:

(12) $k > 1$

The induction hypothesis is:

$$(13)\quad \mathcal{S}^{k'}_{\text{push}-}(v) = \mathcal{R}_{p \,\in\, \{p \,\mid\, p \in \text{Paths}(v) \,\wedge\, C(p) \,\wedge\, \text{length}(p) < k'\}} \mathcal{F}(p) \qquad \text{for all } v \text{ and } k' \leq k$$

We should show that

$$\mathcal{S}^{k+1}_{\text{push}-}(v) = \mathcal{R}_{p \,\in\, \{p \,\mid\, p \in \text{Paths}(v) \,\wedge\, C(p) \,\wedge\, \text{length}(p) < k+1\}} \mathcal{F}(p)$$

By Lemma 15 on [1], [2], [3], [6] and [9], we have

$$(14)\quad \mathcal{S}^{k+1}_{\text{push}-}(v) = \mathcal{R}\left[\mathcal{I}(v),\ \mathcal{R}_{u \in \text{preds}(v)}\left[\mathcal{P}(\mathcal{S}^{k}_{\text{push}-}(u), \langle u, v\rangle)\right]\right]$$

From [14] and [13]

$$\begin{aligned}(15)\quad \mathcal{S}^{k+1}_{\text{push}-}(v) = \mathcal{R}[& \\ & \mathcal{I}(v), \\ & \mathcal{R}_{u \in \text{preds}(v)}\left[\mathcal{P}\left(\mathcal{R}_{p \,\in\, \{p \,\mid\, p \in \text{Paths}(u) \,\wedge\, C(p) \,\wedge\, \text{length}(p) < k\}} \mathcal{F}(p),\ \langle u, v\rangle\right)\right]]\end{aligned}$$

In the case that the size of the set of paths is more than one, from [7], and
in the case that the set of paths is singleton, from $\mathcal{R}_{\{v\}} = v$ and [8], and
in the case that the set of paths is empty, from $\mathcal{R}_{\emptyset} = \bot$ and [6],
we have

$$\begin{aligned}(16)\quad & \mathcal{P}\left(\mathcal{R}_{p \,\in\, \{p \,\mid\, p \in \text{Paths}(u) \,\wedge\, C(p) \,\wedge\, \text{length}(p) < k\}} \mathcal{F}(p),\ \langle u, v\rangle\right) = \\ & \quad \mathcal{R}_{p \,\in\, \{p \,\mid\, p \in \text{Paths}(u) \,\wedge\, C(p) \,\wedge\, \text{length}(p) < k\}} \mathcal{F}(p \cdot \langle u, v\rangle)\end{aligned}$$

After substituting [16] in [15]

$$\begin{aligned}(17)\quad \mathcal{S}^{k+1}_{\text{push}-}(v) = \mathcal{R}[& \\ & \mathcal{I}(v), \\ & \mathcal{R}_{u \in \text{preds}(v)}\left[\mathcal{R}_{p \,\in\, \{p \,\mid\, p \in \text{Paths}(u) \,\wedge\, C(p) \,\wedge\, \text{length}(p) < k\}} \mathcal{F}(p \cdot \langle u, v\rangle)\right]]\end{aligned}$$

that is

$$\begin{aligned}(18)\quad \mathcal{S}^{k+1}_{\text{push}-}(v) = \mathcal{R}[& \\ & \mathcal{I}(v), \\ & \mathcal{R}_{p \,\in\, \{p \,\mid\, \exists u.\ p \in \text{Paths}(u) \,\wedge\, u \in \text{preds}(v) \,\wedge\, C(p) \,\wedge\, \text{length}(p) < k\}} \mathcal{F}(p \cdot \langle u, v\rangle)]\end{aligned}$$

From [18] and Lemma 9

$$\begin{aligned}(19)\quad \mathcal{S}^{k+1}_{\text{push}-}(v) = \mathcal{R}[& \\ & \mathcal{I}(v), \\ & \mathcal{R}_{p \,\in\, \{p \,\mid\, \exists u.\ p \in \text{Paths}(u) \,\wedge\, u \in \text{preds}(v) \,\wedge\, C(p \cdot \langle u,v\rangle) \,\wedge\, \text{length}(p) < k\}} \mathcal{F}(p \cdot \langle u, v\rangle)]\end{aligned}$$

that is

$$\begin{aligned}(20)\quad \mathcal{S}^{k+1}_{\text{push}-}(v) = \mathcal{R}[& \\ & \mathcal{I}(v), \\ & \mathcal{R}_{p' \,\in\, \{p' \,\mid\, p' \in \text{Paths}(v) \,\wedge\, C(p') \,\wedge\, 0 < \text{length}(p') < k+1\}} \mathcal{F}(p')]\end{aligned}$$

From [4] and [5],

$$(21)\quad \mathcal{R}_{p \,\in\, \{p \,\mid\, p \in \text{Paths}(v) \,\wedge\, C(p) \,\wedge\, \text{length}(p) = 0\}} \mathcal{F}(p) = \mathcal{I}(v)$$

From [20] and [10],

$$\begin{aligned}(22)\quad \mathcal{S}^{k+1}_{\text{push}+}(v) = \mathcal{R}[& \\ & \mathcal{R}_{p \,\in\, \{p \,\mid\, p \in \text{Paths}(v) \,\wedge\, C(p) \,\wedge\, \text{length}(p) = 0\}} \mathcal{F}(p), \\ & \mathcal{R}_{p \,\in\, \{p \,\mid\, p \in \text{Paths}(v) \,\wedge\, C(p) \,\wedge\, 0 < \text{length}(p) < k+1\}} \mathcal{F}(p)]\end{aligned}$$

that is

$$\mathcal{S}^{k+1}_{\text{push}+}(v) = \mathcal{R}_{p \,\in\, \{p \,\mid\, p \in \text{Paths}(v) \,\wedge\, C(p) \,\wedge\, \text{length}(p) < k+1\}} \mathcal{F}(p),$$

Lemma 15.

For all $\mathcal{R}, \mathcal{F}, C, \mathcal{I}, \mathcal{P}, k \geq 1$ *if the conditions* $\mathbb{C}_1$ - $\mathbb{C}_9$ *hold,*

$$\mathcal{S}^k_{\text{push}-}(v) = \mathcal{R}\left[\mathcal{I}(v),\ \mathcal{R}_{u \in \mathsf{preds}(v)}\left[\mathcal{P}(\mathcal{S}^{k-1}_{\text{push}-}(u), \langle u, v\rangle)\right]\right]$$

Proof.

We assume that

(1) $\forall n.\ \mathcal{R}(n, \bot) = n$

(2) $\forall n, n'.\ \mathcal{R}(n, n') = \mathcal{R}(n', n)$

(3) $\forall n, n', n''.\ \mathcal{R}(\mathcal{R}(n, n'), n'') = \mathcal{R}(n, \mathcal{R}(n', n''))$

(4) $\forall e.\ \mathcal{P}(\bot, e) = \bot$

(5) $\forall n, n'.\ \mathcal{R}(n, \mathcal{R}(\mathcal{P}(n', \langle u, v\rangle),$
$\mathcal{B}(n', \langle u, v\rangle))) = n$

Proof by induction on k:

Base Case:

(6) $k = 1$

By Def. 4,

(7) $\forall u.\ \mathcal{S}^0_{\text{push}-}(u) = \bot$

(8) $\forall u.\ \mathcal{S}^1_{\text{push}-}(u) = \mathcal{I}(u)$

From [7], [4] and [1],

(9) $\mathcal{R}\left[\mathcal{I}(v),\ \mathcal{R}_{u \in \mathsf{preds}(v)}\left[\mathcal{P}(\mathcal{S}^0_{\text{push}-}(u), \langle u, v\rangle)\right]\right] = \mathcal{I}(v)$

From [9] and [8],

$$\mathcal{S}^1_{\text{push}-}(u) = \mathcal{R}\left[\mathcal{I}(v),\ \mathcal{R}_{u \in \mathsf{preds}(v)}\left[\mathcal{P}(\mathcal{S}^0_{\text{push}-}(u), \langle u, v\rangle)\right]\right]$$

Inductive Case:

The induction hypothesis is

(10) $\mathcal{S}^k_{\text{push}-}(v) = \mathcal{R}\left[\mathcal{I}(v),\ \mathcal{R}_{u \in \mathsf{preds}(v)}\left[\mathcal{P}(\mathcal{S}^{k-1}_{\text{push}-}(u), \langle u, v\rangle)\right]\right]$ forall $k' \leq k$

We show that

$$\mathcal{S}^{k+1}_{\text{push}-}(v) = \mathcal{R}(\mathcal{I}(v), \mathcal{R}_{u \in \mathsf{preds}(v)}\left[\mathcal{P}(\mathcal{S}^k_{\text{push}-}(u), \langle u, v\rangle)\right])$$

From Def. 4, we have that

(11) $\mathcal{S}^{k+1}_{\text{push}-}(v) := S_n$

(12) $\{u_1, .., u_n\} = u \in \left\{u \mid u \in \mathsf{preds}(v) \ \wedge\ \mathcal{S}^k_{\text{push}-}(u) \neq \mathcal{S}^{k-1}_{\text{push}-}(u)\right\}$

(13) $S_0 := \mathcal{S}^k_{\text{push}-}(v)$

(14) $S_{i+1} := \mathcal{R}(\mathcal{R}(S_i,$
$\mathcal{B}\left(\mathcal{S}^{k-1}_{\text{push}-}(u_i), \langle u_i, v\rangle\right))$
$\mathcal{P}\left(\mathcal{S}^k_{\text{push}-}(u_i), \langle u_i, v\rangle\right)$

From [11]-[14], and [2] and [3], we have

(15) $\mathcal{S}^{k+1}_{\text{push}+}(v) = \mathcal{R}(\mathcal{S}^k_{\text{push}-}(v),$
$\mathcal{R}_{u \in \{u \mid u \in \mathsf{preds}(v) \wedge \mathcal{S}^k_{\text{push}-}(u) \neq \mathcal{S}^{k-1}_{\text{push}-}(u)\}}\mathcal{R}($
$\mathcal{B}\left[\mathcal{S}^{k-1}_{\text{push}-}(u_i), \langle u_i, v\rangle\right]$
$\mathcal{P}\left[\mathcal{S}^k_{\text{push}-}(u_i), \langle u_i, v\rangle\right])$

From [15] and [10], we have

$$
\begin{aligned}
(16)\quad \mathcal{S}^{k+1}_{\text{push+}}(v) = \mathcal{R}(\mathcal{R}&\left[\mathcal{I}(v),\ \mathcal{R}_{u\in\text{preds}(v)}\left[\mathcal{P}(\mathcal{S}^{k-1}_{\text{push}-}(u),\langle u,v\rangle)\right]\right],\\
&\mathcal{R}_{u\,\in\,\{u \,\mid\, u\in\text{preds}(v)\,\wedge\,\mathcal{S}^{k}_{\text{push}-}(u)\neq\mathcal{S}^{k-1}_{\text{push}-}(u)\}}\mathcal{R}(\\
&\quad\mathcal{B}\left[\mathcal{S}^{k-1}_{\text{push}-}(u_i),\langle u_i,v\rangle\right]\\
&\quad\mathcal{P}\left[\mathcal{S}^{k}_{\text{push}-}(u_i),\langle u_i,v\rangle\right])
\end{aligned}
$$

that is

$$
\begin{aligned}
(17)\quad \mathcal{S}^{k+1}_{\text{push+}}(v) = \mathcal{R}(\mathcal{R}(\mathcal{I}(v),\mathcal{R}(&\\
&\mathcal{R}_{u\,\in\,\{u \,\mid\, u\in\text{preds}(v)\,\wedge\,\mathcal{S}^{k}_{\text{push}-}(u)=\mathcal{S}^{k-1}_{\text{push}-}(u)\}}\left[\mathcal{P}(\mathcal{S}^{k-1}_{\text{push}-}(u),\langle u,v\rangle)\right],\\
&\mathcal{R}_{u\,\in\,\{u \,\mid\, u\in\text{preds}(v)\,\wedge\,\mathcal{S}^{k}_{\text{push}-}(u)\neq\mathcal{S}^{k-1}_{\text{push}-}(u)\}}\left[\mathcal{P}(\mathcal{S}^{k-1}_{\text{push}-}(u),\langle u,v\rangle)\right]))\\
&\mathcal{R}_{u\,\in\,\{u \,\mid\, u\in\text{preds}(v)\,\wedge\,\mathcal{S}^{k}_{\text{push}-}(u)\neq\mathcal{S}^{k-1}_{\text{push}-}(u)\}}\mathcal{R}(\\
&\quad\mathcal{B}\left[\mathcal{S}^{k-1}_{\text{push}-}(u_i),\langle u_i,v\rangle\right]\\
&\quad\mathcal{P}\left[\mathcal{S}^{k}_{\text{push}-}(u_i),\langle u_i,v\rangle\right])
\end{aligned}
$$

that by [5] is

$$
\begin{aligned}
(18)\quad \mathcal{S}^{k+1}_{\text{push+}}(v) = \mathcal{R}(\mathcal{I}(v),\mathcal{R}(&\\
&\mathcal{R}_{u\,\in\,\{u \,\mid\, u\in\text{preds}(v)\,\wedge\,\mathcal{S}^{k}_{\text{push}-}(u)=\mathcal{S}^{k-1}_{\text{push}-}(u)\}}\left[\mathcal{P}(\mathcal{S}^{k-1}_{\text{push}-}(u),\langle u,v\rangle)\right],\\
&\mathcal{R}_{u\,\in\,\{u \,\mid\, u\in\text{preds}(v)\,\wedge\,\mathcal{S}^{k}_{\text{push}-}(u)\neq\mathcal{S}^{k-1}_{\text{push}-}(u)\}}\ \mathcal{P}\left[\mathcal{S}^{k}_{\text{push}-}(u_i),\langle u_i,v\rangle\right]))
\end{aligned}
$$

that is

$$
\begin{aligned}
(19)\quad \mathcal{S}^{k+1}_{\text{push+}}(v) = \mathcal{R}(\mathcal{I}(v),\mathcal{R}(&\\
&\mathcal{R}_{u\,\in\,\{u \,\mid\, u\in\text{preds}(v)\,\wedge\,\mathcal{S}^{k}_{\text{push}-}(u)=\mathcal{S}^{k-1}_{\text{push}-}(u)\}}\left[\mathcal{P}(\mathcal{S}^{k}_{\text{push}-}(u),\langle u,v\rangle)\right],\\
&\mathcal{R}_{u\,\in\,\{u \,\mid\, u\in\text{preds}(v)\,\wedge\,\mathcal{S}^{k}_{\text{push}-}(u)\neq\mathcal{S}^{k-1}_{\text{push}-}(u)\}}\ \mathcal{P}\left[\mathcal{S}^{k}_{\text{push}-}(u_i),\langle u_i,v\rangle\right]))
\end{aligned}
$$

that is

$$
\begin{aligned}
(20)\quad \mathcal{S}^{k+1}_{\text{push+}}(v) = \mathcal{R}(\mathcal{I}(v),&\\
&\mathcal{R}_{u\in\text{preds}(v)}\left[\mathcal{P}(\mathcal{S}^{k}_{\text{push}-}(u),\langle u,v\rangle)\right])
\end{aligned}
$$

4.4.5 *Termination*

Theorem 27 (Termination).
For all $\mathcal{R}$, $\mathcal{F}$, and C,
if the graph is acyclic or the condition $\mathbb{C}_{10}$ holds, then there exists k such that for every $k' \geq k$
$\mathcal{S}pec^{k'}(v) = \mathcal{S}pec(v)$.

Proof.

We assume that

(1) $\mathcal{S}pec(v) = \mathcal{R}_{p \,\in\, \{p \,|\, p \in \mathsf{Paths}(v) \,\wedge\, C(p)\}}\ \mathcal{F}(p)$

(2) $\mathcal{S}pec^{k}(v) = \mathcal{R}_{p \,\in\, \{p \,|\, p \in \mathsf{Paths}(v) \,\wedge\, C(p) \,\wedge\, \mathsf{length}(p) < k\}}\ \mathcal{F}(p)$

(3) The graph is acyclic or
$\mathbb{C}_{10}: \mathcal{R}(\mathcal{F}(p), \mathcal{F}(\mathsf{simple}(p))) = \mathcal{F}(\mathsf{simple}(p))$

Let

(4) l be the longest simple path to v (that satisfies C).

Let

(5) $P^{l+1} \ = \ \{p \,|\, p \in \mathsf{Paths}(v) \ \wedge \ C(p) \ \wedge \ \mathsf{length}(p) < l+1\}$

(6) $P^{l+i} \ = \ \{p \,|\, p \in \mathsf{Paths}(v) \ \wedge \ C(p) \ \wedge \ \mathsf{length}(p) < l+i\}, \quad i > 1$

(7) $P \ = \ \{p \,|\, p \in \mathsf{Paths}(v) \ \wedge \ C(p)\}$

From [2], [5] and [6], we have

(8) $\mathcal{S}pec(v) = \mathcal{R}_{P}\ \mathcal{F}(p)$

(9) $\mathcal{S}pec^{l+1}(v) = \mathcal{R}_{P^{l+1}}\ \mathcal{F}(p)$

(10) $\mathcal{S}pec^{l+i}(v) = \mathcal{R}_{P^{l+i}}\ \mathcal{F}(p)$

From [4], [7], and [5],

(11) No path in $P \setminus P^{l+1}$ is simple.

(12) No path in $P \setminus P^{l+i}$ is simple.

From [3], we consider two cases:

Case:

(13) The graph is acyclic.

From [11], [12] and [13], we have

(14) $P^{l+1} = P^{l+i} = P$

Thus, from [8], [9] and [10], for $k' = l+1$, for all $k' \geq k$, we have

$\mathcal{S}pec^{k'}(v) = \mathcal{S}pec(v)$

Case:

(15) $\forall p.\, \mathcal{R}(\mathcal{F}(p), \mathcal{F}(\mathsf{simple}(p))) = \mathcal{F}(\mathsf{simple}(p))$

From [11] and [4], we have

(16) $\forall p.\, p \in P \setminus P^{l+1} \longrightarrow \mathsf{length}(\mathsf{simple}(p_1)) < l+1$

From [7], we have

(17) $\forall p.\, p \in P \setminus P^{l+1} \longrightarrow p \in \mathsf{Paths}(v)$

(18) $\forall p.\, p \in P \setminus P^{l+1} \longrightarrow C(p)$

From [17], we have

(19) $\forall p.\, p \in P \setminus P^{l+1} \longrightarrow \mathsf{simple}(p) \in \mathsf{Paths}(v)$

By Lemma 16 and [18],

(20) $\forall p.\, p \in P \setminus P^{l+1} \longrightarrow C(\mathsf{simple}(p))$

From [19], [20], [16] and [5]

(21) $\forall p.\, p \in P \setminus P^{l+1} \longrightarrow \mathsf{simple}(p) \in P^{l+1}$

From [15],

(22) $\forall p.\ p \in P \setminus P^{l+1} \to \mathcal{R}(\mathcal{F}(p), \mathcal{F}(\mathsf{simple}(p))) = \mathcal{F}(\mathsf{simple}(p))$

From [21] and [22],

(23) $\forall p.\ p \in P \setminus P^{l+1} \to \mathcal{R}(\mathcal{F}(p), \mathcal{R}_{P^{l+1}}\ \mathcal{F}(p)) = \mathcal{R}_{P^{l+1}}\ \mathcal{F}(p)$

therefore

(24) $\mathcal{R}(\mathcal{R}_{P \setminus P^{l+1}}\ \mathcal{F}(p), \mathcal{R}_{P^{l+1}}\ \mathcal{F}(p)) = \mathcal{R}_{P^{l+1}}\ \mathcal{F}(p)$

that is

(25) $\mathcal{R}_P\ \mathcal{F}(p) = \mathcal{R}_{P^{l+1}}\ \mathcal{F}(p)$

From [25], [8] and [9], we have

(26) $\mathcal{S}pec(v) = \mathcal{S}pec^{l+1}(v)$

Similarly, for every $k > l + 1$, we can prove that

(27) $\mathcal{S}pec^{k}(v) = \mathcal{S}pec^{l+1}(v)$

From [26] and [27], we have that for $k' \geq l + 1$,

$\mathcal{S}pec^{k'}(v) = \mathcal{S}pec(v)$

Lemma 16.
$\forall p.\ C(p) \leftrightarrow C(\mathsf{simple}(p))$

Proof.
We consider the two cases:
Case:

(1) $C(p) = (head(p) = s)$

Simplification removes cycles but does not change the source vertex, therefore,

$head(p) = s \leftrightarrow head(p \cdot \langle u, v \rangle) = s$

(2) $C(p) = \mathsf{True}$

Straightforward by

$\mathsf{True} \leftrightarrow \mathsf{True}$

5 Implementation

5.1 Mapping Iteration-Map-Reduce to Graph Frameworks

In this section, we map our synthesized functions to graph computations on different graph processing frameworks. We first present the runtime for each framework to understand how different user-defined functions get invoked in these frameworks, and then show how `init_vertex`, `reduce` and `propagate` get utilized for path computations on these frameworks. We select four different graph processing frameworks: PowerGraph [1] and Gemini [4] are distributed graph processing systems, Ligra [2] is a shared-memory graph processing system while GridGraph [5] is a disk-based out-of-core graph processing system. Since these frameworks are highly parallel, we will also discuss how transaction semantics get maintained by our `reduce`.

We note that Gemini, GridGraph and PowerGraph do not inherently support non-idempotent functions. However, all these frameworks can be used to calculate non-idempotent reductions by converting them into idempotent reductions. For example, for the NSP use-case, the non-idempotent sum function can be expressed as a “differential sum” which aggregates only the change in the value instead of the entire new value.

```
class Engine<graph, gather_reducer,
   message_reducer> {
 void run() {
  active: Set of signaled vertices
  next_active = ∅;
  while(active != ∅) {
    par_for(v ∈ signalled) {
     init(v, msg);
     dir_type gd = gather_edges(v);
     par_for(e ∈ edges(v, gd))
      gv = gather(v, e);
     apply(v, gv);
     dir_type sd = scatter_edges(v);
     par_for(e ∈ edges(v, sd))
      scatter(v, e);
    } }
   active = next_active;
   next_active = ∅;
 } };
```

```
int main() {
 Graph<vertex_type, edge_type> g;
 g.load();
 g.transform_vertices(initialize);
 g.transform_edges(init_edge);

 Engine engine = new Engine<g,
      gather_reducer,
      message_reducer>

 engine.map_reduce_edges(signal_vertices);
 // engine.signal_all();
 engine.run();
 // T aggregated_value =
 //   engine.map_reduce_vertices<T>(transform);
}
```

Fig. 22. PowerGraph Runtime

5.2 PowerGraph

PowerGraph is a distributed graph processing system that provides a shared-memory programming abstraction. It efficiently processes power-law graphs by incorporating a vertex-cut strategy for balanced workload distribution, and by parallelizing vertex computations across edges. It achieves this by splitting vertex-computations across three steps: `gather`, `apply`, and `scatter`. Figure 22 shows PowerGraph's iterative processing model. The `run()` method processes a set of vertices in each iteration by invoking five functions (marked in blue). The `gather()` function iterates through edges of a vertex (incoming, outgoing, both or none, as defined by `gather_edges()`) to aggregate the values from its neighbors. The `apply()` function computes a new value of the vertex based on the aggregated value from the gather step. Finally, the `scatter()` function iterates through edges of a vertex (incoming, outgoing, both or none, as defined by `scatter_edges()`) to propagate its new value to its neighbors.

In each iteration, the set of vertices to be processed are identified via explicit vertex-signaling mechanism. Typically, if a vertex's value changes, it 'signals' its neighbors in the `scatter()` function so that they get processed in the subsequent iteration. For the first iteration, the set of vertices to be processed are signalled before invoking the `run()` method (as shown in `main()`).

Apart from iterative processing, PowerGraph also provides capabilities for transforming and reducing vertex (and edge) values. The `map_reduce_vertices()` function shown in `main()` can be used to perform vertex-based reductions.

Mapping Synthesized Functions.

PowerGraph allows expressing graph computations in pull mode (Figure 23) and in push mode (Figure 24). In pull mode, the propagation of values across edges occurs in the `gather` step, and the values propagated to a vertex (or 'pulled by a vertex') in this step are passed through an aggregator as defined in `struct reducer`. In push mode, value propagation occurs in the `scatter` step and

```
struct reducer {
 VValueType value;
 reducer& operator+=(reducer& other) {
  value = reduce(value, other.value);
  return *this;
 } }

bool changed = false;
void init(vertex_type& v, empty_type& m) { }

dir_type gather_edges(vertex_type& v) {
 return in_edges; }

reducer gather(vertex_type& v, edge_type& e) {
 if (e.source().data() != none) {
  return propagate(e.source(), e);
 } else {
  return none;
 } }
```

```
void apply(vertex_type& v, reducer& red_gv) {
 changed = false;
 if(reduce(red_gv.value, v.data()) !=
           v.data()) {
  v.data() = red_gv.value;
  changed = true;
 }
}

dir_type scatter_edges(vertex_type& v) {
 return changed ? out_edges : no_edges;
}

void scatter(vertex_type& v,
             edge_type& e) {
 signal(e.target());
}
```

Fig. 23. PowerGraph Pull

```
struct reducer {
 VValueType value;
 reducer& operator+=(reducer& other) {
  value = reduce(value, other.value);
  return *this;
 } }

bool changed = false;
reducer msg;
void init(vertex_type& v,
          empty_type& m) {
 msg = m;
}

dir_type gather_edges(vertex_type& v) {
 return no_edges;
}

reducer gather(vertex_type& v,
               edge_type& e) { }
```

```
void apply(vertex_type& v,
           reducer& red_gv) {
 changed = false;
 if(reduce(msg.value, v.data()) !=
           v.data()) {
  v.data() = msg.value;
  changed = true;
 } }

dir_type scatter_edges(vertex_type& v) {
 return changed ? out_edges : no_edges;
}

void scatter(vertex_type& v,
             edge_type& e) {
 VValueType new_val = propagate(v, e);
 if(reduce(new_val, e.target().data()) !=
           e.target().data()) {
  signal(e.target(), new_val);
 } }
```

Fig. 24. PowerGraph Push

the values propagated to a vertex (or ‘pushed to a vertex’) in this step are passed through the aggregator.

In both the modes, the aggregated value is again passed to `reduce()` operation along with the vertex’s current value to identify whether the aggregated value is useful. Due to monotonic nature of `reduce()`, the usefulness of the value is directly determined by `!=` operator. It is interesting to note that push mode can eliminate unnecessary value propagations by invoking `reduce()` on the neighboring vertex during `scatter` to check usefulness of the value before propagating. Also, since PowerGraph's semantics ensure that the entire vertex program (`gather-apply-scatter`) gets executed atomically, we synthesize `reduce()` using simple (non-atomic) operators.

Finally, vertex initializations are achieved via a map operation on vertices (by `transform_vertices()` operation in `main()` function). Furthermore, vertex-based reduction is achieved by passing two functions to `map_reduce_vertices()`: an aggregation function that performs reduction, and a transformation function that updates vertex values before aggregation.

5.3 Ligra

Ligra is a single machine shared memory graph processing system that parallelizes computations across edges and vertices. Since our path-based computations wholly operate on edges, we show Ligra's `edgeMap()` operation in Figure 25. Given a subset of vertices `U` and an edge function `f()`, the `edgeMap` applies `f()` on all the outgoing edges of vertices in `U`. It is interesting to note that edge function `f()` must maintain atomicity.

Mapping Synthesized Functions.

Since `edgeMap` operates on outgoing edges, we compute our path algorithms in push mode. As shown in Figure 25, our `compute()` method iteratively invokes `edgeMap()` on `frontier` vertices, i.e., those whose values have been updated. The initial vertex frontier can be defined as the source vertex for computations relying on the source, or as the entire vertex set when source is not available (e.g., for connected components algorithm).

Figure 25 shows the structure of our edge function. It propagates value from source to destination and immediately reduces the propagated value with the destination's current value. The reduction operation writes the new value for destination vertex if the propagated value is better than destination's current value. It is important to note that Ligra invokes edge operations concurrently without atomicity guarantees like PowerGraph. To maintain atomicity in our `edgeFunction()`, our `reduce()` operation writes the final value using `CAS` operation.

While Ligra does not natively provide aggregation over vertices, we implemented a parallel vertex aggregator that maps over vertices and aggregates their values to perform vertex-based reductions.

```
vertexSubset edgeMap(graph g,
     vertexSubset U, func f, func c) {
 vertexSubset out = ∅;
 par_for(v ∈ U)
  par_for(ngh ∈ out_neighbors(v))
   if(c(ngh) && f(v, ngh, w(ngh)))
    out = out.insert(ngh);
 return out;
}

void compute(graph g) {
 VValueType* values =
       new VValueType[g.n];
 par_for(VIdType i=0;i<n;i++)
  values[i] = initialize(i);
 vertexSubset frontier(n,src);
 // vertexSubset frontier(n, n,
 //              [1, 1, .., 1]);
```

```
 while(!frontier.isEmpty()) {
  next_frontier = edgeMap(g,
     frontier, edgeFunction,
     condFunction);
  frontier.del();
  frontier = next_frontier;
 }
 frontier.del();
}

bool edgeFunction(VIdType s, VIdType d,
                    EWeightType w) {
 return reduce(&values[d],
        propagate(s, EdgeType(s, d, w)));
}

bool condFunction(VIdType d)
 { return true; }
```

Fig. 25. Ligra Runtime & Push

5.4 Graphit

Graphit is a single machine shared memory graph processing DSL and framework that parallelizes computations across edges and vertices Graphit utilizes different scheduling models. GRAFS has adopted the push scheduling model shown in the 26. Given a frontier `U` and an struct type containing the edge function `f()`, the `edgeMap` applies `f()` on all the outgoing edges of vertices in `U`. It is interesting to note that edge function `f()` must maintain atomicity.

Mapping Synthesized Functions.

As shown in Figure 26, the `main()` method iteratively invokes `edgeMap()` on `frontier` vertices, i.e., those whose values have been updated. The initial vertex frontier can be defined as the source vertex for computations relying on the source, or as the entire vertex set when source is not available (e.g., for connected components algorithm).

Figure 26 `edgeMap()` shows the structure of our edge function. It propagates value from source to destination and immediately reduces the propagated value with the destination's current value. The reduction operation writes the new value for destination vertex if the propagated value is better than destination's current value. It is important to note that Graphit invokes edge operations concurrently without atomicity guarantees like PowerGraph. To maintain atomicity in the `edgeMap()`, the `reduce()` operation writes the final value using `CAS` operation. To support map and reduce over the vertices, we have adopted parallel for structure in Graphit framework.

```
template<typename EDGE_MAP>
vertexSubset edgeset_apply(WGraph g,
      vertexSubset U, EDGE_MAP f) {
 vertexSubset out = ∅;
 par_for(v ∈ U)
  par_for(ngh ∈ out_neighbors(v))
   if(f(v, ngh, w(ngh)))
    out = out.insert(ngh);
 return out;
}

struct edgeMap {
 bool operator(NodeID s, NodeID d, int w) {
  return reduce(&values[d],
         propagate(s, EdgeType(s, d, w)));
 }
}
```

```
int main() {
 WGraph g;
 g.load();
 VValueType* values =
        new VValueType[g.n];
 par_for(VIdType i=0;i<n;i++)
  values[i] = initialize(i);
 vertexSubset frontier(n,src);
 //vertexSubset frontier(n,n);
 addVertex(frontier, src) ;
 while(!frontier.isEmpty()) {
  next_frontier =
    edgeset_apply(edges, frontier, edgeMap());
  frontier.del();
  frontier = next_frontier;
 }
 frontier.del();
}
```

Fig. 26. Graphit Runtime & Push

5.5 Gemini

Gemini is a NUMA-aware, distributed, high-performance graph processing system. It extracts parallelism across multicores by partitioning threads across NUMA nodes, and uses MPI for coordination across machines. It incorporates a hybrid push-pull processing model that dynamically switches between pull mode and push mode depending on the number of active vertices. The pull mode is performed when number of active vertices is large (based on a threshold), and it effectively iterates over all the incoming edges of a vertex to compute its next value. On the other hand, the push mode is performed when number of active vertices is small and it iterates over all the outgoing edges of a vertex to compute their next value.

Similar to Ligra, we show `process_edges()` in Figure 27 since our path-based computations operate on edges only. As we can see, `process_edges()` accepts four user-defined callbacks along with a bitmask indicating the set of active vertices. The bitmask is first checked to determine sparsity of the iteration, based on which, either the first two callbacks are invoked (if sparse), or the other two call backs are invoked (if dense). The `sparse_signal` and `dense_signal` callbacks determine the value to be propagated from/to a vertex to/from its outgoing and incoming neighbors respectively. These values are maintained in form of messages, that are shuffled and sorted across NUMA nodes and machines. Then, the `sparse_slot` and `dense_slot` callbacks compute the new vertex value based on the propagated values (or messages) from `sparse_signal` and `dense_signal` respectively, and also activate neighboring vertices to be processed in the next iteration. It is interesting to note that iterating over the incoming and outgoing edges is performed by the user-defined callbacks, as opposed to the runtime as achieved in PowerGraph and Ligra.

Mapping Synthesized Functions.

We leverage Gemini's hybrid push-pull processing model by expressing our path-based computations in both, push mode and pull mode. The `main()` method in Figure 27 first activates the source vertex by setting its bit value, and then iteratively calls `process_edges()` (setting all bits activates all vertices, as required by algorithms like connected components).

In push mode (`sparse_signal` and `sparse_slot`), the source vertex emits its value which is propagated to the outgoing neighbors. Similarly, in the pull mode (`dense_signal` and `dense_slot`), the destination propagates in the values from its incoming neighbors using which it computes the best value for itself. To ensure atomicity, similar to that for Ligra, `CAS` operation is used to write the final value in `reduce()`. Vertex-based reductions are also achieved in same manner as in Ligra.

```
VertexId process_edges(func sparse_signal,
  func sparse_slot, func dense_signal,
  func dense_slot, Bitmap* active) {
 sparse = compute_sparsity(active);
 if(sparse) {
  par_for(VertexId v ∈ active)
   sparse_signal(v);
  exchange_messages();
  par_for(msg ∈ messages) {
   VertexId source = message.vertex;
   sparse_slot(source, message.msg_data, outAdjList[v]);
  }
 } else {
  par_for(VertexId v ∈ V)
   dense_signal(v, inAdjList[v]);
  exchange_messages();
  par_for(msg ∈ messages) {
   VertexId target = message.vertex;
   dense_slot(target, message.msg_data);
  }
 }
}

int main() {
 Graph g;
 g.load();
 values = g->alloc_vertex_array<VValueType>();
 VertexSubset* active_in = g->alloc_vertex_subset();
 VertexSubset* active_out = g->alloc_vertex_subset();

 for(VertexId i=0; i<g->vertices; ++i)
  values[i] = initialize(i);

 active_in->clear();
 active_in->set_bit(src);
 VertexId num_active_vertices = 1;
 // active_in->fill();
 // VertexId num_active_vertices = graph->vertices;
```

```
while(num_active_vertices > 0) {
 active_out->clear();
 num_active_vertices = g->process_edges(
  [&](VertexId src){
   g->emit(src, values[src]);
  },
  [&](VertexId src, VValueType msg, AdjList out_nbrs) {
   VertexId activated = 0;
   for (AdjUnit* ptr ∈ out_nbrs) {
    VertexId dst = ptr->neighbour;
    if(reduce(&values[dst], propagate(msg,
               EdgeType(src, dst, ptr->edge_data)))) {
     active_out->set_bit(dst);
     activated += 1;
    }
   }
   return activated;
  },
  [&](VertexId dst, AdjList in_nbrs) {
   VValueType msg = none;
   for (AdjUnit* ptr ∈ in_nbrs) {
    VertexId src = ptr->neighbour;
    reduce(&msg, propagate(values[src],
            EdgeType(src, dst, ptr->edge_data)));
   }
   if (msg != none) g->emit(dst, msg);
  },
  [&](VertexId dst, VValueType msg) {
   if(reduce(&values[dst], msg)) {
    active_out->set_bit(dst);
    return 1;
   }
   return 0;
  },
  active_in
 );
 swap(active_in, active_out);
}
}
```

Fig. 27. Gemini Hybrid Push-Pull Runtime

5.6 GridGraph

GridGraph is an out-of-core disk-based graph processing system. It maintains the graph in a 2D grid layout that resides on disk, and uses a streaming partition based processing model to sequentially accesses disk partitions. Figure 28 shows `stream_edges` and `stream_vertices` that are used to process the graph. The `stream_edges` function processes active set of edges by reading the corresponding partitions from disk one-by-one and invoking the user-defined `process` function on the edge. The `stream_vertices` function invokes a user-defined function on active vertices (similar to MAP operation). It is interesting to note that both these methods take care of disk operations so that the user-defined functions can focus solely on edge and vertex computations.

Mapping Synthesized Functions.

Similar to Ligra, we express our path computations on GridGraph in push mode. The `main` function first initializes the vertex values using `stream_vertices`, after which it iteratively calls `stream_edges` on outgoing edges of active vertices. For each edge, the computation propagates the source's value to the destination in parallel (CAS operation used in `reduce()` for atomicity).

```
void stream_edges(func process, Bitmap* active) {
 for(partition p ∈ partitions) {
  if(p ∉ active)
   continue;
  for(Edge e ∈ p)
   if(e.source ∈ active)
    process(e);
 }
}

void stream_vertices(func process, Bitmap* active) {
 par_for(VertexId v ∈ V) {
  if(v ∈ active)
   process(v);
 }
}

int main() {
 Graph g(load_path);
 Bitmap* active_in = g.alloc_bitmap();
 Bitmap* active_out = g.alloc_bitmap();
 vertex_values.init(vertex_path, g.vertices);

 g.stream_vertices<VertexId>([&](VertexId i) {
  vertex_values[i] = initialize(i);
  return 0;
 });

 active_out->clear();
 active_out->set_bit(src);
 VertexId num_active_vertices = 1;
 // active_out->fill();
 // VertexId num_active_vertices = g.vertices;

 while (num_active_vertices > 0) {
  swap(active_in, active_out);
  active_out->clear();
  active_vertices = g.stream_edges<VertexId>([&]
             (Edge& e) {
   if (reduce(&vertex_values[e.target],
      propagate(
         e.source,
         EdgeType(e.source, e.target, e.w))
      )) {
    active_out->set_bit(e.target);
    return 1;
   }
   return 0;
  }, active_in);
 } }
```

Fig. 28. GridGraph Runtime

5.7 Path-based Reduction Synthesis

```
//NWR usecase
struct VValueType{
 uint32_t first;
 uint32_t second;
};
VValueType reduce(const VValueType a,
  const VValueType b) {
 bool r = 0;
 VValueType c;
 VValueType w;
 do {
  c = a;
  w = c;
  if (b.first < c.first) {
   w.first = b.first;
  } else {
   if (b.first > c.first) {
    w.first = c.first;
   }

  }if (b.second > c.second) {
   w.second = b.second;

  } else {
   if (b.second < c.second) {
    w.second = c.second;
   }
  }
 } while(((b.second > c.second ||
   b.first < c.first) &&
     !(r=cas(a,c,w))));
 return r;
}
```

```
//Radius usecase
struct VValueType{
 uint32_t first;
 uint32_t second;
};
VValueType reduce(const VValueType a,
  const VValueType b) {
 bool r = 0;
 VValueType c;
 VValueType w;
 do {
  c = a;
  w = c;
  if (b.first < c.first) {
   w.first = b.first;
  } else {
   if (b.first > c.first) {
    w.first = c.first;
   }
  }if (b.second < c.second) {
   w.second = b.second;
  } else {
   if (b.second > c.second) {
    w.second = c.second;
   }
  }
 } while(((b.second < c.second ||
     b.first < c.first) &&
     !(r=cas(a,c,w))));
 return r;
}
```

Fig. 29. Generated atomic reduce functions for more elaborate use-cases. The rule FMPair is used to generate atomic reduce functions for NWR and Radius use-cases, respectively.

```
//BFS usecase
struct VValueType{
 uint32_t first;
 uint32_t second;
};
VValueType reduce(const VValueType a,
  const VValueType b) {
 bool r = 0;
 VValueType c;
 VValueType w;
 do {
  c = a;
  w = c;
  if (b.first < c.first) {
   w.first = b.first;
   w.second = b.second;
  } else {
   if (b.first > c.first) {
    w.first = c.first;
    w.second = c.second;
   }
  }
 } while((b.first < c.first &&
  !(r=cas(a,c,w)))); return r;
}
```

```
//CC usecase
struct VValueType{
 uint32_t first;
};
VValueType reduce(const VValueType a,
  const VValueType b) {
 bool r = 0;
 VValueType c;
 VValueType w;
 do {
  c = a;
  w = c;
  if (b.first > c.first) {
   w.first = b.first;
  } else {
   if (b.first < c.first) {
    w.first = c.first;
   }
  }
 } while((b.first > c.first &&
  !(r=cas(a,c,w)))); return r;
}
```

```
//SP usecase
struct VValueType{
 uint32_t first;
};
VValueType reduce(const VValueType a,
  const VValueType b) {
 bool r = 0;
 VValueType c;
 VValueType w;
 do {
  c = a;
  w = c;
  if (b.first < c.first) {
   w.first = b.first;
  } else {
   if (b.first > c.first) {
    w.first = c.first;
   }
  }
 } while((b.first < c.first &&
  !(r=cas(a,c,w)))); return r;
}
```

```
//WP usecase
struct VValueType{
 uint32_t first;
};
VValueType reduce(const VValueType a,
  const VValueType b) {
 bool r = 0;
 VValueType c;
 VValueType w;
 do {
  c = a;
  w = c;
  if (b.first > c.first) {
   w.first = b.first;
  } else {
   if (b.first < c.first) {
    w.first = c.first;
   }
  }
 } while((b.first > c.first &&
  !(r=cas(a,c,w)))); return r;
}
```

Fig. 30. Generated atomic reduce functions for simple use-cases.

```
//WSP usecase
struct VValueType{
 uint32_t first;
 uint32_t second;
};

VValueType reduce(const VValueType a,
  const VValueType b) {
 bool r = 0;
 VValueType c;
 VValueType w;
 do {
  c = a;
  w = c;
  if (b.first < c.first) {
   w.first = b.first;
   w.second = b.second;
  } else {
   if (b.first > c.first) {
    w.first = c.first;
    w.second = c.second;
   }

  }if (c.first == b.first) {
   w.first = c.first;
   w.second = std::max(b.second, c.second);
  }

 } while(((b.first < c.first ||
   (c.first == b.first &&
   b.second > c.second)) &&
   !(r=cas(a,c,w))));
 return r;
}
```

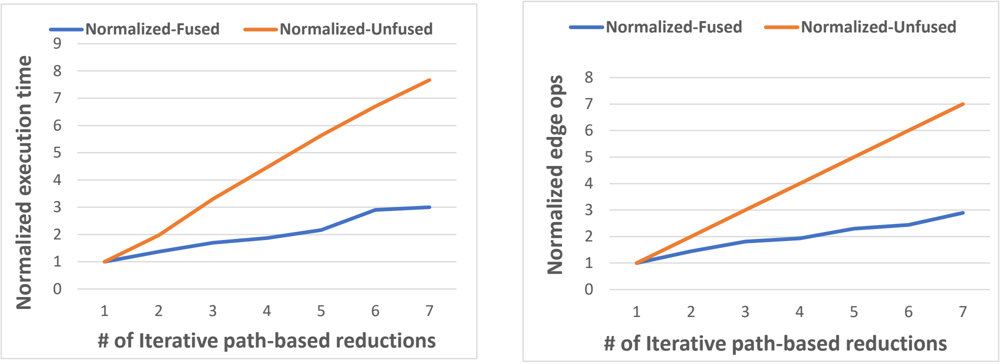
Fig. 31. Generated atomic reduce function for WSP usecase. The rule FPNest is used to generate atomic reduce functions for WSP usecase.

6 Experimental Results

We presented the core of our experimental results in the main body of the paper. We present the rest of the experimental results in this section.

- In § 6.1, we study the scalability of fusion. We measure the speedup as the number of fusions increase.
- In § 6.2 we report the weighted graphs execution times for the unweighted graph execution times reported in the main body of the paper § 7, Fig. 15.
- In § 6.3, we report the execution times for the normalized execution times reported in the main body of the paper § 7, Fig. 16.
- In § 6.4, we compare the performance of the push, pull and the hybrid models.
- In § 6.5, we compare the synthesized and handwritten programs for streaming graphs.

6.1 Fusion Scalability



(a) (b)

Fig. 32. Fusion scalability of Grafs on the Radius use-case. The graph is unweighted LiveJournal. The backend is PowerGraph. (a) Normalized execution time with respect to the execution time of one path-based reduction and (b) Normalized number of edge operations with respect to the number of edge operations for one path-based reduction.

In this section, we study the scalability of the fusion transformations. We show that the performance of the synthesized code scales with the number of fusions.

We compare the fused and unfused implementations of the Radius use-case over several sample sizes. We increase the size of the sample source set from 1 to 7. Thus, the number of path-based reductions is increased from 1 to 7. Accordingly, the number of possible fusions is increased from 1 to 7 as well. We run the experiment on the unweighted LiveJournal graph in PowerGraph (push model) framework. Fig. 32 presents the results that are normalized with respect to the Radius instance with sample size of one i.e. one path-based reduction.

Fig. 32a shows the execution time of both fused and unfused implementations of the code, normalized with respect to the execution time of one path-based reduction. Fig. 32b shows the number of processed edges in both fused and unfused implementations, normalized with respect to number of edges processed by one path-based reduction. With the increase in the sample size, we observe a linear increase in the execution time and processed edges for the unfused implementation. The reason for the linear increase is that the unfused implementation performs the iterative computations for the sources separately. However, the fused implementation benefits from the overlapping computations in each iteration and performs them together. Hence, it results in a faster execution time and a fewer number of edge operations. Thus, it exhibits more scalability than the unfused implementation.

We note that fusion might be beneficial up to a limit on the number of fused operations. Fusing many values into a tuple may lead to memory overheads and affect performance due to lack of locality. A cost model can automatically determine whether fusion can improve performance, and the granularity of fusion. The cost model can be developed by profiling the dynamic behavior of the queries on the input graphs.

6.2 The Effect of Fusion

Table 3. Execution times (in seconds). H: Handwritten, S: Synthesized, R: the ratio $\frac{H}{S}$.

Prog.	Input	Ligra			GridGraph			Gemini			PowerGraph (Push)			PowerGraph (Pull)			GraphIt (Push)		
		S	H	**R**	S	H	**R**	S	H	**R**	S	H	**R**	S	H	**R**	S	H	**R**
DRR	LJ	1.2	2.7	**2.3**	3.7	16.3	**4.3**	0.5	1.4	**2.8**	9.4	31.7	**3.3**	16.5	60	**3.6**	0.75	2.2	**2.9**
	TW	-	-	-	82	215	**2.6**	7	16	**2.2**	61	184	**3**	107	392	**3.6**	12	41	**3.3**
	TM	-	-	-	130	325	**2.5**	33	110	**3.3**	94	313	**3.3**	223	760	**3.4**	202	345	**1.7**
	FR	-	-	-	223	464	**2**	27	68	**2.5**	202	520	**2.5**	297	1093	**3.6**	-	-	-
Trust	LJ	1.1	2.6	**2.3**	6.2	16	**2.5**	0.7	1.32	**1.8**	-	-	-	19.7	54	**2.7**	1	2.2	**2.1**
	TW	-	-	-	2413	2433	**1**	11.9	16	**1.3**	-	-	-	151	392	**2.6**	23	48	**2.1**
	TM	-	-	-	3215	5312	**1.6**	24	18	**0.75**	-	-	-	214	636	**3**	940	370	**0.4**
	FR	-	-	-	540	620	**1.1**	7965	11105	**1.4**	364	419	**1.1**	367	1003	**2.7**	-	-	-
LTrust	LJ	1.7	2.2	**1.4**	6.7	10	**1.5**	0.8	1.2	**1.4**	23	33	**1.4**	-	-	-	1.3	2.2	**1.7**
	TW	-	-	-	86	168	**1.9**	10	15.3	**1.5**	150	193	**1.2**	-	-	-	25	41	**1.6**
	TM	-	-	-	142	186	**1.3**	12	16.2	**1.3**	281	324	**1.1**	-	-	-	324	679	**2.1**
	FR	-	-	-	584	1048	**1.8**	5300	7315	**1.3**	389	442	**1.2**	-	-	-	-	-	-

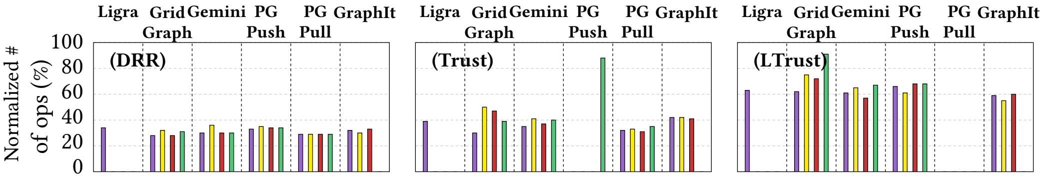


Fig. 33. Edge-work Ratio: Normalized # of edges processed by the fused over the unfused version. Missing bars are due time-out after 24 hours.

Here we present the results for the effect of fusion on more elaborated use-cases. Similar to Fig. 15 and Table 1 in section § 7, we report The edge-work ratio and absolute execution times for weighted graphs in Fig. 33 and Table 3 respectively. We can observe that like unweighted graphs, fusing results in overal 2.1× speedup across different frameworks and input graphs.

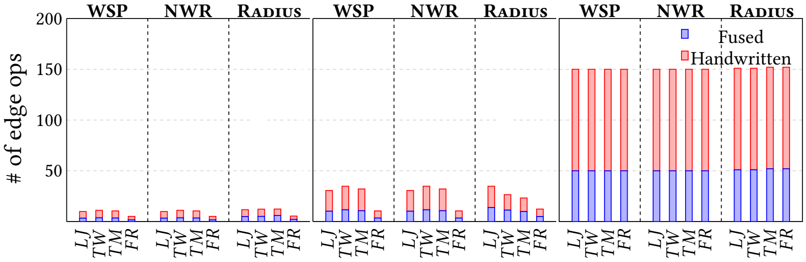
6.3 Fusion Types

Use-case	Input	Ligra			GridGraph			Gemini			PowerGraph (Push)			PowerGraph (Pull)		
		H	S	**R**	H	S	**R**	H	S	**R**	H	S	**R**	H	S	**R**
WSP	LJ	1	0.7	1.4	5.3	3.2	1.65	0.54	0.4	1.35	7	3.2	2.1	18.3	8.9	2
	TW				37.4	19.6	1.9	10	6.5	1.5	47.2	27	1.74	130.9	69.6	1.9
	TM				71	37.4	1.9	14.3	9.3	1.5	78.7	45.3	1.7	199.2	92.5	2.1
	FR				142.6	81	1.7	15.7	9.9	1.5	116.1	59.6	1.9	237.5	116.8	2
Radius	LJ	1.3	0.9	1.4	7.3	3.2	2.2	1.6	1.18	1.45	7.1	4.1	1.73	19	11.7	1.6
	TW				40.8	21	1.9	38.6	24.6	1.6	55	45.7	1.2	156	80.6	1.9
	TM				70.2	35.6	1.9	66.6	39.6	1.6	84.3	67.7	1.2	237	130.6	1.8
	FR				151.4	89.4	1.7	218.2	104.2	2	115	75.9	1.5	234	126	1.8
NWR	LJ	0.9	1.2	1.3	4	2.9	1.4	0.6	0.4	1.4	7.8	3.6	2.1	17.7	8	2.2
	TW				37.8	20.8	1.8	14.4	6.7	2.1	52.7	23.4	2.2	132.7	63	2.1
	TM				62	41	1.5	22	11	2	76	38.1	2	200	97.1	2
	FR				134.4	72.4	1.8	22.6	10.5	2.1	116.2	63.6	1.8	226.9	115.1	1.9

Table 2. Execution times in seconds of the fused and unfused implementations. (H: Handwritten, F: Synthesized, R=$\frac{H}{S}$). Missing cells are due to out of memory executions.

In order to study the performance benefits of the different fusion types that the fusion rules represent, in § 7, we studied the three use-cases WSP, NWR and Radius (from Fig. 6). In § 7, Fig. 16, we compared the number of edges processed by the synthesized fused programs with that by the unfused versions for unweighted graphs. Here, we compare the execution time of the synthesized programs with that of the unfused versions for weighted graphs. Table 2 presents the execution times of both synthesized and handwritten implementations along with the speedup of the synthesized implementations over the handwritten implementations that is the execution time of the later divided by the former. In spite of variances across different input graphs and different frameworks, as expected, synthesized implementations benefiting from fusion rules can execute faster than the handwritten versions. Fusion results in an overall speedup of 1.4-2.1×.

6.4 Gemini Framework Analysis



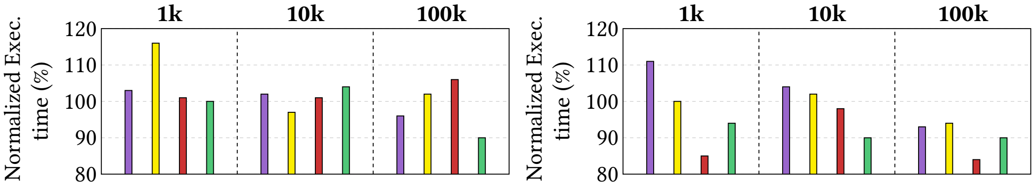
(a) Push (b) Hybrid (c) Pull

Fig. 34. Normalized number of edge operations in Gemini framework

We compared the performance of the push, pull and hybrid models on the Gemini framework. Fig. 34 presents the number of edge operations that each of the WSP, NWR and Radius use-cases process for each of the input graphs separately for each of the push, pull and hybrid models. The number of processed edges for each use-case and input graph is normalized with respect to the number of edges that the use-case processes on that input graph in the unfused implementation with the pull model. We observe that overall, the push model is more efficient than the hybrid model and the hybrid model is more efficient than the pull model. Similar to Fig. 16 in the main body § 7, we also observe again that the fused versions process about 50% less edges than the unfused versions.

6.5 Streaming Evaluation

In this section we present the evaluation of dynamic graphs with edge mutations. Fig. 35 shows the normalized execution time of the handwritten implementation in KickStarter framework [3] with respect to the synthesized code for the same framework on the GRAFS for SSSP and CC use-cases. We also report the absolute execution times in Table 3. The experiments show that GRAFS can effectively synthesize streaming use-cases that run on dynamic graphs and match the performance of the handwritten implementations in the KickStarter framework.



(a) SSSP

(b) CC

Fig. 35. Normalized execution time of the handwritten implementation in KickStarter framework with respect to the synthesized version in the GRAFS for a) SSSP and b) CC use-cases on dynamic input graphs with 1k, 10k and 100k edge mutations.

# Edge Mutations	Use-case	LJ		TW		TM		FR	
		H	S	H	S	H	S	H	S
1*k*	SSSP	0.0056	0.0054	0.0186	0.016	0.0182	0.0179	0.0311	0.031
10*k*		0.0095	0.0093	0.0223	0.0229	0.0270	0.0265	0.0401	0.0383
100*k*		0.0253	0.0263	0.032	0.0312	0.036	0.0339	0.0716	0.0792
1*k*	CC	0.004	0.0036	0.0123	0.0123	0.0148	0.0174	0.0216	0.0229
10*k*		0.006	0.006	0.0185	0.018	0.0224	0.0228	0.0348	0.0387
100*k*		0.0156	0.0166	0.0244	0.0258	0.0282	0.0338	0.0493	0.0549

Table 3. Execution times in seconds of the synthesized and handwritten implementations. (H: Handwritten, S: Synthesized)

$$\begin{array}{lll}
\multicolumn{3}{l}{\textsc{PageRank}\ (\mathrm{PR})} \\
\mathcal{I} & := & \lambda v.\ \ 1\,/\,|V| \\
\mathcal{P} & := & \lambda n, e.\ \ n\,/\,\mathrm{outdeg}(\mathrm{src}(e)) \\
\mathcal{R} & := & \lambda v, v'.\ \ v + v' \\
\mathcal{E} & := & \lambda n.\ \ \gamma * n \ + \ (1-\gamma)\,/\,|V| \\
\mathcal{B} & := & \lambda n, e.\ \ -\mathcal{E}^{-1}(n)\,/\,\mathrm{outdeg}(\mathrm{src}(e))
\end{array}$$

Fig. 36. Optimized PageRank Use-case using Def. 7. $\mathcal{E}^{-1}(n)$ denotes the inverse of the $\mathcal{E}$ function. Note that the back propagation ($\mathcal{B}$) is calculated starting from the second iteration.

References

[1] Joseph E Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. Powergraph: Distributed graph-parallel computation on natural graphs. In *Presented as part of the 10th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 12)*, pages 17–30, 2012.

[2] Julian Shun and Guy E. Blelloch. Ligra: A lightweight graph processing framework for shared memory. In *Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '13, pages 135–146, New York, NY, USA, 2013. ACM.

[3] Keval Vora, Rajiv Gupta, and Guoqing Xu. KickStarter: Fast and Accurate Computations on Streaming Graphs via Trimmed Approximations. pages 237–251, 2017.

[4] Xiaowei Zhu, Wenguang Chen, Weimin Zheng, and Xiaosong Ma. Gemini: A computation-centric distributed graph processing system. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 301–316, 2016.

[5] Xiaowei Zhu, Wentao Han, and Wenguang Chen. Gridgraph: Large-scale graph processing on a single machine using 2-level hierarchical partitioning. In *2015 {USENIX} Annual Technical Conference ({USENIX}{ATC} 15)*, pages 375–386, 2015.