GRAFS: Graph Analytics Fusion and Synthesis Appendix

ANONYMOUS AUTHOR(S)

Contents

9	Cont	tents	
10	1	Use-case Specifications	1
11	2	Specification and Fusion	8
12	2.1	Semantics	8
13	2.2	Language and Fusion Extensions	10
14	2.2.1	Common Operation Elimination	11
15	2.2.2	Domain	12
16	2.2.3	Unary operations and Literals	12
17	2.2.4	Vertex Variables	13
18	2.2.5	Syntactic Sugar	15
19	2.2.6	Nested Triple-lets	17
20	2.3	Example Fusions	20
21	3	Mapping Specification to Iteration-Map-Reduce	22
22	3.1	Iterative Reduction and its Correctness	22
23	3.1.1	Pull Model	22
24	3.1.2	Push Model	24
25	3.1.3	Asynchronous Model	28
26	3.1.4	Streaming Graphs	31
27	3.1.5	Factored Path-based Reductions	33
28	3.2	Synthesis of Iterative Reduction	34
29	4	Proofs	37
30	4.1	Helper Definitions	37
31	4.2	Semantics Compositionality	38
32	4.3	Soundness of Fusion	40
33	4.4	Iteration Correctness Conditions	45
34	4.4.1	Pull, Idempotent	45
35	4.4.2	Pull, Non-idempotent	49
36	4.4.3	Push, Idempotent	53
37	4.4.4	Push, Non-idempotent	57
38	4.4.5	Termination	64
39	5	Implementation	66
40	5.1	Mapping Iteration-Map-Reduce to Graph Frameworks	66
41	5.2	PowerGraph	67
42	5.3	Ligra	70
43	5.4	Graphit	71
44	5.5	Gemini	72
45	5.6	GridGraph	74
46	5.7	Path-based Reduction Synthesis	75
47	6	Experimental Results	78
48	6.1	Fusion Scalability	79

Anon.

50	6.2	The Effect of Fusion	80
51	6.3	Fusion Types	81
52	6.4	Gemini Framework Analysis	82
53	6.5	Streaming Evaluation	83
54	Refer	rences	85
55			
56			
57			
58			

- - 85 86
 - 87 88

99	1	Use-case Specific	cati	ons	
100		SSSP(s)(v)	=	min weight(p)	Shortest Path
101				$p \in Paths(s,v)$	
102					
103		NP(s)(v)	=	$ \operatorname{Paths}(s,v) $	Number of Paths
104		IP(s)(n)	_	max weight(h)	Longest Path
105		$\operatorname{Li}(3)(0)$	_	$p \in Paths(s,v)$	Longest I ath
106		/ \ / \			
107		SL(s)(v)	=	$\min_{\substack{p \in Paths(s,v)}} length(p)$	Shortest Length
108				r(-,)	
109		LL(s)(v)	=	\max length(p)	Longest Length
110				$p \in Paths(s,v)$	
111		WP(s)(n)	_	max $canacity(p)$	Widest Path
112		W1(3)(0)	_	$p \in \text{Paths}(s, v)$	wheest I attr
113					
114		NP(s)(v)	=	$\min_{p \in Paths(s, p)} capacity(p)$	Narrowest Path
115				$p \in ratio(s, b)$	
116		FR(s)(n)	_		Forward Reachability
117		II(3)(0)	_	$\phi \in Paths(s, v)$	101 ward Reachability
118				p cratis(3,0)	
119		CC(v)	=	min head(p)	Connected Components
120				$p \in Paths(v)$	1
121		/ >			
122		CCS(v)	=	$\bigcup \{ \operatorname{head}(p) \}$	Connected Component Set
123				$p \in Paths(v)$	
124		BP(c)(n)	_		Backword Passhability
125		DR(3)(0)	-	\bigvee frue $p \in Paths(n,s)$	Dackwaru Keachability
126				$p \in Iattis(0,3)$	
127		BFS(s)(v)	=	penultimate($\arg \min$ length(p))	Breadth-First Search
128				$p \in Paths(s,v)$	
129					
130					
131				Fig. 1. Use-cases for $\mathcal{R}_{p \in P} f(p)$ and $f(x)$	$\arg \mathcal{R} f(p)$
132				P C	$p \in P$
133					
134					
135					
136					
137					
138					
139					
140					
141					
142					
1/2					
144					
144					
140					

148 149	WSP(s)(v)	=	let $P := \underset{p \in Paths(s,v)}{\operatorname{args min}} \operatorname{length}(p)$ in	Widest Shortest Paths
150			$\max_{p \in P} capacity(p)$	
151				
152	NSP(s)(n)	_	$args min_weight(n)$	Number of Shortest Paths
153	101 (3)(0)	-	$p \in Paths(s,v)$	Number of Shortest Faths
154				
155	HLP(v)	=	head($\arg \max weight(p)$)	Head of Longest Path
156			$p \in Paths(v)$	
157	HLL(v)	=	head($\arg \max$ length(p))	Head of Longest Length
158			$p \in Paths(v)$	
159	$\operatorname{UND}(n)$	_	haad(argmin conscitu(b))	Head of Norrowast Dath
161	$\operatorname{IIINF}(0)$	-	$p \in Paths(v)$	fiead of Natiowest Fath
162				
163	SWSL(s)(v)	=		Shortest Weight in
164			Let D	Shortest Length Paths
165			let $P := \arg s \min \operatorname{lengtn}(p) \ln p \in \operatorname{Paths}(s, v)$	
166			min weight(p)	
167			$p \in P$	
168				Widest in
169	WSLSW(s)(v)	=		Shortest Length in
170				Shortest Weight Paths
171			let $P := \underset{p \in P}{\operatorname{args min}} \operatorname{weight}(p)$ in	
172			let $P' := \arg \min \operatorname{length}(p)$ in	
173			$p \in P$	
174			$\max_{p \in P'} \operatorname{capacity}(p)$	
175				
170	LNP(s)(v)	=		Longest Narrowest Path
178			let $P := \underset{p \in Paths(s, p)}{\operatorname{argsmin}} \operatorname{capacity}(p)$ in	
179			max length(p)	
180			$p \in P$	
181	HNP(v)	=		Heads of Narrowest Paths
182			$P \coloneqq \arg \min \operatorname{capacity}(p)$ in	
183			$p \in Paths(v)$	
184			$\bigcup \{ \operatorname{head}(p) \}$	
185			$p \in P$	
186			1	
187	CCSS(v)	=	$\left \right \left \left\{ \text{head}(p) \right\} \right $	Connected Component Set Size
188			$p \in Paths(v)$	×
189				
190			Fig. 2. Use-cases for nested $\mathcal{R}_{p \in P} f$	(<i>p</i>), part 1
191			<i>p</i> ⊂i	
192				
194				
195				
196				

Number of Widest Shortest Paths NWSP(s)(v) =let $P := \operatorname{args\,min} \operatorname{weight}(p)$ in $p \in Paths(s,v)$ let $P' \coloneqq \operatorname{args} \max \operatorname{capacity}(p)$ $p \in P$ |P'|Number of NSWSL(s)(v) =Shortest Weight in Shortest Length Paths let $P := \operatorname{args\,min} \operatorname{length}(p)$ in $p \in Paths(s,v)$ let $P' := \operatorname{args} \min \operatorname{weight}(p')$ in $p \in P$ | P' | Fig. 3. Use-cases for nested $\underset{p \in P}{\mathcal{R}} f(p)$, part 2

Anon.

246 247 248	NWR(s)(v)	=	$\frac{\min_{\substack{p \in \text{Paths}(s,v) \\ p \in \text{Paths}(s,v)}} \text{capacity}(p)}{\max_{\substack{p \in \text{Paths}(s,v)}} \text{capacity}(p)}$	Narrowest to Widest Path Ratio
249 250 251 252	LSD(s)(v)	=	LP(s)(v) - SSSP(s)(v) max $p \in Paths(s,v)$ weight(p) - min $p \in Paths(s,v)$ weight(p)	Difference between Longest and Shortest Path
253 254 255 256 257	SP2(s,s')(v)	=	$\min (SSSP(s)(v), SSSP(s')(v))$ $\min \left(\min_{p \in Paths(s,v)} weight(p), \min_{p \in Paths(s',v)} weight(p) \right)$	Shortest Path from Two Sources
258 259 260 261 262 263	SPR(s, s')(v)	=	$\frac{\frac{\text{SSSP}(s)(v)}{\text{SSSP}(s')(v)}}{\underset{p \in \text{Paths}(s,v)}{\min} \text{weight}(p)}$ $\frac{\max_{p \in \text{Paths}(s',v)} \text{weight}(p)}{\underset{p \in \text{Paths}(s',v)}{\max} \text{weight}(p)}$	Ratio of Shortest Paths from Two Sources
264 265 266 267 268			Fig. 4. Use-cases for nested $m \oplus m$	
269 270 271 272 273 274				
275 276 277 278 279				
280 281 282 283 284				
285 286 287 288 289				
289 290 291 292				

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

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

$$Ecc(s) = \min_{v \in V} WP(s)(v) \qquad \text{The Capacity of the Narrowest of the Widest Paths from s to All Vertices}$$

$$= \min_{v \in V} \max_{p \in Paths(s, v)} capacity(p)$$

$$ILNPG(s) = \max_{v \in V} LNP(s)(v) \qquad \text{The Length of the Longest of the Narrowest Paths from s to All Vertices}$$

$$= |e| P(v) := \arg_{p \in Paths(s, v)} capacity(p) in$$

$$\max_{v \in V} \max_{p \in Paths(s, v)} ecc(v)| \qquad \text{Number of Connected Components}$$

$$= \left| \bigcup_{v \in V} (Cc(v) \right| \qquad \text{Number of Connected Components}$$

$$= \left| \bigcup_{v \in V} ecp(s) = \exp_{v \in V} ecc(v) \right|$$

$$RFA = \bigcap_{v \in V} CC(s) \qquad \text{Vertices Reachable to All Vertices}$$

$$= \bigcap_{v \in V} \bigcup_{p \in Paths(s, v)} ecc(v) = ecc(v) = ecc(v)$$

$$Fig. 5. Use-cases for R m ecc(v) = ecc(v)$$

Anon.

344	Radius	=	$\min \max \min length(p)$	Radius Sampled on vertices $\{\overline{v}\}$
345			$s \in \{\overline{v}\} \ v \in V \ p \in Paths(s,v)$	
346	Diam	=	max max min length(p)	Diameter Sampled on vertices $\{\overline{n}\}$
347			$s \in \{\overline{v}\} \ v \in V \ p \in Paths(s,v)$	
348			DIAM	Diameter to Radius Ratio
349	DRR	=	RADIUS	
350			RADIUS	
351			$\max_{s \in \{\overline{n}\}} \max_{p \in Paths(s, p)} \operatorname{length}(p)$	
352		=	$\frac{1}{1} \frac{1}{1} \frac{1}$	
353			$s \in \{\overline{v}\} \ v \in V \ p \in Paths(s,v)$	
354				
355	BC(s)	=	let S := $\lambda s, v. \min_{p \in Paths(s,v)} length(p)$ in	
356				
357			let $N := \lambda s, v$. args $\min_{p \in \text{Paths}(s,v)} \text{length}(p) $ In	
358			$\sum \qquad N(v)(s) \times N(s)(t)$	
359			$v \neq t \in V \land$	
360			$\sum \frac{S(v)(s) + S(s)(t) = S(v)(t)}{\sum}$	
361			$\sum_{v \in \overline{s}} N(v, t)$	
362			$v \neq t \in V$	
363				
364			Fig. 7. Use-cases for $r \oplus$	r
365	BC specifi	ies t	he betweenness centrality algorithm from	a sampled set of nodes \overline{s} . For every
366	pair of node	s (so	ource is from sampled set and destination i	s over all the nodes), it calculates the
367	number of sh	orte	st paths that goes through s. The nominator	calculates the number of sortest paths
368	(N) from v to	ottł	nat passes through s. It uses a vertex-based	reduction constrained by path-based

³⁶⁸ (N) from v to t that passes through s. It uses a vertex-based reduction constrained by path-based ³⁶⁹ reductions similar to DS. Similarly, the denominator calculates all the shortest paths from v to ³⁷⁰ t. Finally, Betweenness Centrality measure is calculated using sum vertex-based reduction over ³⁷¹ sampled nodes.

$$NPH(s)(v) = \sum_{p \in Paths(s,v)} length(p) \mapsto 1$$
Number of Paths Histogram
$$ISP(s)(v) = \sum_{p' \in args min \\ p \in Paths(s,v)} veright(p)$$

$$CCH(u) = \sum_{v \in V} (\min_{p \in Paths(v)} head(p)) \mapsto 1$$
Connected Components Sizes
$$Fig. 8. Use-cases with map values$$

2 Specification and Fusion

⁴⁴³ 2.1 Semantics

SMBin $\left[\!\!\!\left[\begin{array}{c} \mathcal{R} \mathcal{F}(p) \\ p \in P \end{array} \right]\!\!\!\left[(g) = \overline{\left[\mathbf{v} \mapsto \mathcal{R} \left\{ \mathcal{F}(p) \mid p \in \left[\!\!\left[P \right]\!\right](g)(\mathbf{v}) \right\} \right]}_{\mathbf{v} \in \mathbf{V}(g)} \right]$ $\llbracket m \oplus m' \rrbracket (q) = \llbracket m \rrbracket (q) \oplus \llbracket m' \rrbracket (q)$ $\begin{bmatrix} \text{VAR} \\ \llbracket x \rrbracket (g) = \bot \end{bmatrix}$ $\left\| \text{ let } X \coloneqq M \text{ in } e \right\|(g) = \overline{\left[\mathbf{v} \mapsto \left[e \left[X \coloneqq \left[M \right] \right](g)(\mathbf{v}) \right] \right]}_{\mathbf{v} \in \mathsf{V}(q)}$ $\begin{bmatrix} \mathcal{R} & m \end{bmatrix} (g) = \mathcal{R} \left\{ \overline{\llbracket m \rrbracket(g)(\mathsf{v})}_{\mathsf{v} \in \mathsf{V}(g)} \right\} \qquad \qquad \begin{array}{c} \text{SRBIN} \\ \llbracket r \oplus r' \rrbracket(g) = \llbracket r \rrbracket(g) \oplus \llbracket r' \rrbracket(g) \end{array}$ $\| e^{\operatorname{Int} X} := M \text{ in} \\ \operatorname{mlet} X' := E \text{ in} \\ \operatorname{rlet} X'' := R \text{ in} \\ g = \left[e \left[X'' := \left[R \left[X' := \left[E \left[X := \left[M \right] \right] (g) \right] \right] \right] \right] (g) \right] \right]$ SPATHS $[\operatorname{Paths}](q) = \overline{[\mathsf{v} \mapsto \{p \mid p \in \operatorname{Paths}(q) \land \operatorname{tail}(p) = \mathsf{v}\}]}_{\mathsf{v} \in \mathsf{V}(q)}$ $\begin{array}{l} \text{SMPAIR} & \text{SMPAIR} \\ \llbracket \langle M, M' \rangle \rrbracket (g) = \left\langle \llbracket M \rrbracket (g), \llbracket M' \rrbracket (g) \right\rangle & \llbracket \mathcal{R} \mathcal{F} \rrbracket = \left[\begin{array}{c} \mathcal{R} \\ \mathcal{R} \mathcal{F} \rrbracket = \left[\begin{array}{c} \mathcal{R} \\ \mathcal{R} \mathcal{F} \\ \mathcal{R} \mathcal{F} \end{bmatrix} \right] & \llbracket \langle \mathbb{R}, \mathbb{R}' \rangle \rrbracket (g) = \left\langle \llbracket \mathbb{R} \rrbracket (g), \llbracket \mathbb{R}' \rrbracket (g) \right\rangle \end{array}$ $\left[\left[\mathcal{R}\left\langle \overline{[\mathbf{v}\mapsto n_{\mathbf{v}}]}_{\mathbf{v}\in\mathsf{V}(g)},...,\overline{[\mathbf{v}\mapsto n_{\mathbf{v}}']}_{\mathbf{v}\in\mathsf{V}(g)}\right\rangle\right]\right] = \left[\left[\mathcal{R}\left\langle \overline{[\mathbf{n}\mapsto\langle n_{\mathbf{v}},...,n_{\mathbf{v}}'\rangle]}_{\mathbf{v}\in\mathsf{V}(g)}\right\rangle\right]\right]$ SEVALSEMSEEPAIR $[\![n]\!] = n$ $[\![d]\!] = d$ $[\![\langle E, E' \rangle]\!] = \langle [\![E]\!], [\![E']\!] \rangle$ SEBIN $\llbracket e \oplus e' \rrbracket = \llbracket e \rrbracket \oplus \llbracket e' \rrbracket$

Fig. 9. Denotational Semantics of the language presented in Fig. 9 of the main paper. The notation $[k_i \mapsto v_i]_i$ represents a finite map that maps each key k_i to value v_i over the range *i*. The notation X := 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 $[\![]\!]$ 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 $V(g) \mapsto \mathbb{N}$ from each vertex of *g* to natural numbers, and \bot (for undefined). The rule SPRED defines the semantics of the path-base reduction $\mathcal{R}_{p \in P} \mathcal{F}(p)$. (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 491 to v. For each vertex v, it applies the function \mathcal{F} to each path to v and then applies the reduction 492 493 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 494 the the semantics of $m \oplus m'$ as the result of the operator \oplus on the semantics of *m* and *m'*. Whether 495 the notations \mathcal{R} and \oplus refer to the syntactic or semantic domains is clear from the context: they are 496 in the syntactic and semantic domains when they are respectively on the left- and right-hand side of 497 498 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 ilet X := M in e as the pointwise substitution 499 of the variables X with the semantics of M in e. Pointwise substitution replaces variables with 500 values from a corresponding pair of structures. (The formal definition of substitution is available in 501 the appendix § 4.1). The rule VAR states that the semantics of free variables is undefined. 502

503 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 \perp . The rule SVRED defines the semantics of the 504 vertex-based reduction \mathcal{R} *m* using the map resulted from the semantics of *m*; it reduces the values 505 506 of the map for all vertices. The rule SRBIN defines the semantics of $r \oplus r'$ as the result of applying 507 the operator \oplus to the semantics of *r* and *r'*. The rule SRLET defines the semantics of triple-let terms 508 by three subsequent substitutions: the substitution of the variables X with the semantics of M in E, 509 the substitution of the variables X' with the semantics of E in R, and finally the substitution of the 510 variables X'' with the semantics of R in e. 511

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 $\arg \mathcal{R} \mathcal{F}(p)$ where \mathcal{R} is min or max using the map resulted $p \in P$

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 $\mathcal{R} \not\in$ as a path-based reduction on the paths Paths. The rule SRR defines the semantics of $\mathcal{R} (\overline{[v \mapsto n_v]}_{v \in V(g)}, ..., \overline{[v \mapsto n'_v]}_{v \in V(g)})$ as a vertex-based reduction on $\overline{\langle n_v, ..., n'_v \rangle}_{v \in V(g)}$. 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.

529 LEMMA 1 (COMPOSITIONALITY). 530 For all r, r' and \mathbb{R} , if $\llbracket r \rrbracket = \llbracket r' \rrbracket$ then $\llbracket \mathbb{R}[r] \rrbracket = \llbracket \mathbb{R}[r'] \rrbracket$. 531 For all m, m', and \mathbb{M} , if $\llbracket m \rrbracket = \llbracket m' \rrbracket$ then $\llbracket \mathbb{M}[m] \rrbracket = \llbracket \mathbb{M}[m'] \rrbracket$. 532 For all M, M', and $\mathbb{M}s$, if $\llbracket M \rrbracket = \llbracket M' \rrbracket$ then $\llbracket \mathbb{M}s[M] \rrbracket = \llbracket \mathbb{M}s[M'] \rrbracket$. 533 For all R, R', and $\mathbb{R}s$, if $\llbracket R \rrbracket = \llbracket R' \rrbracket$ then $\llbracket \mathbb{R}s[R] \rrbracket = \llbracket \mathbb{R}s[R'] \rrbracket$. 534 536 537

-
- 538
- 539

540 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.

543				
544	r	:=	$\left \begin{array}{c} \mathcal{R} m \oplus r \\ \mathcal{V} \end{array} \right \left \begin{array}{c} \mathcal{R} m \\ \mathcal{R} m \end{array} \right \left r \oplus r \right \circ r \left n \right $	Vertex-based Reduction
545				
546			ilet $X \coloneqq M$ in	
547			mlet $X \coloneqq E$ in	
548			rlet $X := R$ in	
549			$e \mid x$	
550	m	:=	$\mathcal{K}_{p \in P} F(p) \mid m \oplus m \mid p \in P$	Path-based Reduction
551			$\boxed{\left \text{ilet } X := M \text{ in } v e \right } \mid x$	
552				
553	Р	:=	Paths(v) Paths(v, v') args $\mathcal{R} F(p)$	Paths
554			<i>p</i> ∈ <i>P</i>	
555	M	:=	$\langle M, M \rangle \mid \left \left \mathcal{R}_{c} \mathcal{F} \right \right $	
556				
557	R	:=	$\dots \mid \left \begin{array}{c} \mathcal{R} \\ \mathcal{V} \\ \mathcal{N} \end{array} \right $	Context for <i>r</i>
558	v			Vertex Variable
559	S	:=	$v \mid \perp$	Source
560	0	:=	\rightarrow \leftarrow	Orientation
561	с	:=	$s o \mid \langle c, c \rangle$	Path Configuration
562	${\mathcal R}$:=	ı U ∩ ı	Reduction Operation
563	Ŧ	•=		Path Function
564)	.–		r atti r anetion

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

589 2.2.1 Common Operation Elimination

590	IFTIM
591	$(\text{ ilet } \langle X_1, X_2 \rangle := \langle \mathcal{R} \mathcal{F}, \mathcal{R} \mathcal{F} \rangle \text{ in }) (\text{ ilet } X_1 := \mathcal{R} \mathcal{F} \text{ in })$
592	mlet $X' := E$ in mlet $X' := E[X_2 \mapsto X_1]$ in
593	rlet $X'' \coloneqq R$ in \Rightarrow rlet $X'' \coloneqq R$ in
594	
595	
596	$\begin{array}{c} \text{ICom} \\ (\text{ilst}/Y, Y) := (M, M) \text{in} \\ \end{array} $
597	$ \left(\begin{array}{c} \operatorname{Ilet} \langle X_1, X_2 \rangle := \langle M_1, M_2 \rangle \operatorname{III} \\ \operatorname{mlet} Y' := F \operatorname{in} \end{array} \right) \left(\begin{array}{c} \operatorname{Ilet} \langle X_2, X_1 \rangle := \langle M_2, M_1 \rangle \operatorname{III} \\ \operatorname{mlet} Y' := F \operatorname{in} \end{array} \right) $
598	H E(X) = E H $ H E(X) = E H $ $ H E(X) = E H $
599	
600	
601	IAssL
602	$(\operatorname{ilet} \langle X_1, \langle X_2, X_3 \rangle) \coloneqq \langle M_1, \langle M_2, M_3 \rangle) \text{ in } (\operatorname{ilet} \langle \langle X_1, X_2 \rangle, X_3 \rangle \coloneqq \langle \langle X_1, X_2 \rangle, M_3 \rangle) \text{ in } (\operatorname{ilet} \langle X_1, X_2 \rangle, X_3 \rangle \coloneqq \langle X_1, X_2 \rangle, M_3 \rangle) $
603	$ \begin{array}{c} \text{mlet } X' \coloneqq E \text{ in} \\ \Rightarrow \\ \text{mlet } Y'' \Rightarrow p \Rightarrow \\ \text{mlet } X'' \Rightarrow p \Rightarrow \\ \end{array} $
604	$\operatorname{riet} X := K \operatorname{In}$
605	(e)) (e))
606	IAssR
607	$(\text{ ilet } \langle \langle X_1, X_2 \rangle, X_3 \rangle \coloneqq \langle \langle M_1, M_2 \rangle, M_3 \rangle \text{ in }) (\text{ ilet } \langle X_1, \langle X_2, X_3 \rangle \rangle \coloneqq \langle M_1, \langle M_2, M_3 \rangle \rangle \text{ in })$
608	$mlet X' \coloneqq E in \\ \Rightarrow mlet X' \coloneqq E in \\ = mlet X' = m$
609	rlet $X'' \coloneqq R$ in rlet $X'' \coloneqq R$ in
610	(e) (e)

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 andthe 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.

638 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 \cup and intersection \cap 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

645 646 2.2.3 Unary operations and Literals

FILIT FMLVAR 647 ilet $x \coloneqq \perp$ in n \Rightarrow ilet $x' \coloneqq \perp$ in xn \Rightarrow x 648 649 FMPAIR" FMPAIR' 650 $\langle X, x \rangle \coloneqq \langle M, \bot \rangle$ $X \coloneqq M$ $\langle x, X \rangle \coloneqq \langle \bot, M \rangle \longrightarrow X \coloneqq M$ \rightarrow 651 652 FRLit ilet $x \coloneqq \perp$ in 653 \Rightarrow n mlet $x' \coloneqq \bot$ in 654 rlet $x'' \coloneqq \perp \text{ in } n$ 655 656 FRPAIR' FRPAIR" 657 $\langle x, X \rangle \coloneqq \langle \bot, R \rangle \longrightarrow X \coloneqq R$ $\langle X, x \rangle := \langle R, \bot \rangle \longrightarrow$ $X \coloneqq R$ 658 659 FRUNI $\circ \begin{pmatrix} \text{ilet } X \coloneqq M \text{ in} \\ \text{mlet } X' \coloneqq E \text{ in} \\ \text{rlet } X'' \coloneqq R \text{ in} \\ e \end{pmatrix} \implies \begin{pmatrix} \text{ilet } X \coloneqq M \text{ in} \\ \text{mlet } X' \coloneqq E \text{ in} \\ \text{rlet } X'' \coloneqq R \text{ in} \\ e e \end{pmatrix}$ 660 FIUni 661 \circ (ilet $X \coloneqq M$ in e) \Rightarrow ilet X := M in $\circ e$ 662 663

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 \circ . 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- \perp reductions, the rules FMPAIR', FMPAIR'', FRPAIR' and FRPAIR'' remove the dummy \perp reductions. The two rules FIUNI and FRUNI simply apply the unary operator \circ to the resulting expression *e*.

12

664

665 666

667

668

669

670

687 2.2.4 Vertex Variables

688 $\begin{array}{ll} \operatorname{FPRED}' & \operatorname{FPRED}'' \\ & \mathcal{R} & F(p) \\ & \Rightarrow_{m} \\ \operatorname{ilet} x \coloneqq \mathcal{R} & \mathcal{F} \operatorname{in} v' x \end{array} & \begin{array}{ll} \operatorname{FPRED}'' \\ & \mathcal{R} & \mathcal{F}(p) \\ & \Rightarrow_{m} \\ & \Rightarrow_{m} \\ \operatorname{ilet} x \coloneqq \mathcal{R} & \mathcal{F} \operatorname{in} v x \end{array}$ $\begin{array}{l} \mathcal{R} \\ \mathcal{R} \\ p \in \mathsf{Paths}(v) \\ \Rightarrow_{m} \\ \mathsf{ilet} \ x \coloneqq \mathcal{R} \\ \mathcal{F} \ \mathsf{in} \ v \ x \end{array}$ 689 690 691 692 693 FILETBIN 694 $(\text{ilet } X_1 \coloneqq M_1 \text{ in } v e_1) \oplus (\text{ilet } X_2 \coloneqq M_2 \text{ in } v e_2)$ if $free(e_1) \cap X_2 = \emptyset$ $free(e_2) \cap X_1 = \emptyset$ 695 \Rightarrow_m 696 ilet $\langle X_1, X_2 \rangle \coloneqq \langle M_1, M_2 \rangle$ in $v (e_1 \oplus e_2)$ 697 698 $\begin{array}{l} \mathcal{R} \\ \mathcal{R} \\ v \in \mathsf{V} \end{array} (\text{ilet } X \coloneqq \mathcal{R}' f \text{ in } v e) \quad \Rightarrow_r \quad \text{ilet } X \coloneqq \mathcal{R}' f \text{ in } \\ c \\ \text{mlet } x \coloneqq e \text{ in} \end{array}$ 699 $\begin{pmatrix} \mathcal{R} \mathcal{F}, \mathcal{R}' \mathcal{F}' \\ {}_{c}^{c} \mathcal{F}, {}_{c'}^{c'} \mathcal{F}' \end{pmatrix} \quad \Rightarrow_{M} \quad \underset{\langle c, c' \rangle}{\mathcal{R}''} \mathcal{F}''$ 700 rlet $x' \coloneqq \mathcal{R} x$ in x'701 where $f'' \coloneqq \lambda p. \langle F'(p), F(p) \rangle$ 702 $\mathcal{R}^{\prime\prime}(\langle a,b\rangle,\langle a^{\prime},b^{\prime}\rangle) \coloneqq$

 $\langle \mathcal{R}(a,a'), \mathcal{R}'(b,b') \rangle$

703 704

705

706

707

708

709

710

711

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 \mathcal{R}_{V} *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 712 vertex variables. Only the changed or new non-terminals are shown and the updated parts are 713 boxed with solid lines. The extended vertex-based reduction \mathcal{R} *m* binds the vertex variable *v*. The 714 $v \in V$ 715 path constructors specify source and destination: the term Paths(v) specifies the set of paths with 716 any source and the destination v and the term Paths(v, v') specifies the set of paths with the source 717 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 718 719 in the destination vertices v'. It can also calculate the reduction over paths from every source 720 vertex v to a destination vertex v' and store the result in the source vertices v. We call the vertex 721 variable where the result is stored, the target vertex. The let constructor ilet X := M in v e of the path-based reductions *m* carries the vertex *v* that stores the result of the factored reduction *M* with 723 the expression *e*.

The source *s* of paths can be either a vertex *v* or none \perp . The orientation *o* of paths is either forward \rightarrow or backward \leftarrow . 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 $\mathcal{R} \mathcal{F}$ 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 Paths(v) to a let term with a factored path-based reduction that has no source \perp , forward orientation \rightarrow , and the target vertex v. The rules FPRED' and FPRED'' both convert a path-based reduction over Paths(v, v') to let forms. The former stores the results in the destination vertices and the latter stores the results in rad

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.

785 2.2.5 Syntactic Sugar 786 FMRed $\mathcal{F}(\underset{p \in P}{\operatorname{ring}} \mathcal{R} \mathcal{F}'(p)) := \operatorname{ilet} \langle x, x' \rangle \coloneqq \underset{p \in P}{\mathcal{R}'} \mathcal{F}''(p) \operatorname{in} x' \text{ where } \mathcal{R} \in \{\min, \max\}$ $\mathcal{F}'' \coloneqq \lambda p. \langle \mathcal{F}'(p), \mathcal{F}(p) \rangle$ 787 788 789 $\mathcal{R}'(\langle a, b \rangle, \langle a', b' \rangle) \coloneqq \text{if } (\mathcal{R}(a, a') = a) \text{ then } \langle a, b \rangle \text{ else } \langle a', b' \rangle$ 790 791 $|P| := \sum_{p \in P} 1$ 792 793 794 ROp $\mathcal{R}_{v \in \{v_1, \dots, v_n\}} m \quad \coloneqq \quad \left(\left(m[v \coloneqq v_1] \mathcal{R} \ m[v \coloneqq v_2] \right) \mathcal{R} \ \dots \ m[v \coloneqq v_n] \right)$ 795 796 797 VSel 798 799 800 where 801 $\mathcal{R}'(\langle a, b \rangle, \langle a', b' \rangle) \coloneqq$ 802 if $(a \wedge a')$ then $\langle a, \mathcal{R}(b, b') \rangle$ 803 else if (a') then $\langle a', b' \rangle$ 804 else $\langle a, b \rangle$ 805 806 Fig. 14. Syntactic Sugar 807 Syntactic sugar enable concise specifications. In Fig. 14, we present the syntactic sugar and the 808 rules that desugar them. 809 The term $\mathcal{F}(\arg \mathcal{R} \mathcal{F}'(p))$ where \mathcal{R} is either min or max first finds a path p in P with the 810 811 minimum or maximum value for the function \mathcal{F}' and then returns the result of applying \mathcal{F} to p. It 812 is used to specify the BFS use-case. The rule FMRED expands this term to a path-based reduction in the let form ilet $\langle x, x' \rangle \coloneqq \underset{p \in P}{\mathcal{R}'} \mathcal{F}''(p)$ in x'. The path function \mathcal{F}'' returns the pair of the results of 813 814 \mathcal{F}' and \mathcal{F} . The reduction function \mathcal{R}' returns the input pair with the minimum or maximum first 815 element. 816 The term |P| specifies the size of the set of paths *P*. It is used to specify the NSP use-case. The 817 rule PSIZE simply expands it to the path-based reduction $\sum_{n=1}^{\infty} 1$ that counts the number of paths. 818 $\mathcal{R}_{v \in \{v_1,..,v_n\}}$ *m* is a vertex-based reduction over a limited set of vertices $\{v_1, .., v_n\}$. It is 819 The term 820 used to specify the RADIUS use-case. The rule ROP expands this term to operations between to 821 path-based reductions $m[v := v_i], i \in \{1..n\}$. The operation corresponds to the reduction function 822 \mathcal{R} ; for example, the reduction function Σ is unrolled to the operation +. 823 The term $\mathcal{R}_{v \in V \land m'}$ *m* specifies a vertex-based reduction of *m* over the selected vertices *v* for which 824 825 m' evaluates to true. This idiom was used to specify the DS use-case. The rule VSEL expands it to the path-based reduction $\mathcal{R}'_{v \in V}(m', m)$. The path-based reduction calculates a pair of values for m'826 827 and *m* at every vertex. Then, the vertex-based reduction \mathcal{R}' only reduces the second elements of 828 the pairs whose first element is true. Given two input pair, the vertex-based reduction \mathcal{R}' applies 829 the reduction $\mathcal R$ to the second elements if the first elements of both pairs are true. Otherwise, the 830 pair whose first element is true is selected. If the first element of both pairs is false, either of them 831 can be selected; this definition selects the first. Finally, in the following if expression, there are two 832 833

Anon.

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.

883 2.2.6 Nested Triple-lets

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

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 $\underset{V}{\mathcal{R}} m \oplus r$ 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 $\underset{V}{\mathcal{R}} m \oplus \mathbb{R}$ 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*.

$$LTRUST(s) = let SSSP := \lambda s, v. \min_{\substack{p \in Paths(s,v) \\ p \in Paths(s,v)}} weight(p) in$$
$$let NP := \lambda s, v. \min_{\substack{p \in Paths(s,v) \\ min \\ v \in V \land SSSP(s,v) < RADIUS}} NP(s,v)$$

Unrolling the let terms results in the following:

$$LTRUST(s) = \min_{v \in V \land \left(\min_{p \in Paths(s,v)} weight(p)\right) < RADIUS} \left(\min_{p \in Paths(s,v)} capacity(p)\right)$$

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

$$LTRUST(s) = \mathcal{R}\left(\left(\min_{p \in Paths(s,v)} weight(p)\right) < RADIUS, \left(\min_{p \in Paths(s,v)} capacity(p)\right)\right)$$

where $\mathcal{R}(b, \langle a', b' \rangle) \coloneqq$

if
$$(a')$$
 then $\min(b, b')$
else b

We note that in the above specification, the path-based reduction SSSP(s, v) < RADIUS includes the nested vertex-based reduction RADIUS. From Fig. 2, RADIUS can be fused to following:

1 ())

$$RADIUS = \begin{pmatrix} \text{ilet } \langle x, y \rangle \coloneqq \mathcal{R}' \not\in \text{in} \\ \langle s_1, s_2 \rangle \\ \text{mlet } \langle x', y' \rangle \coloneqq \langle x, y \rangle \text{ in} \\ \text{rlet } \langle x'', y'' \rangle \coloneqq \mathcal{R}'' \langle x', y' \rangle \text{ in} \\ \min(x'', y'') \end{pmatrix} \xrightarrow{\mathcal{F}} \left\{ \begin{array}{c} \mathcal{P} \colon [\operatorname{Apt}(p), [\operatorname{engtn}(p)) \\ \mathcal{R}' (\langle a, b \rangle, \langle a', b' \rangle) \coloneqq \\ \min(a, a'), \min(b, b') \rangle \\ \mathcal{R}'' (\langle a, b \rangle, \langle a', b' \rangle) \coloneqq \\ (\operatorname{max}(a, a'), \operatorname{max}(b, b') \rangle \\ \end{array} \right\}$$

 σ

1 /1

.1 () 1

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:

$$LT_{RUST}(s) = let \ radius := \begin{pmatrix} ilet \langle x, y \rangle \coloneqq \mathcal{R}' \ \mathcal{F} \text{ in} \\ mlet \langle x', y' \rangle \coloneqq \langle x, y \rangle \text{ in} \\ rlet \langle x'', y'' \rangle \coloneqq \mathcal{R}'' \langle x', y' \rangle \text{ in} \\ min(x'', y'') \end{pmatrix} \text{ in} \\ \mathcal{R} \ \left(\begin{pmatrix} \min_{p \in Paths(s,v)} \text{ weight}(p) \end{pmatrix} < radius, \begin{pmatrix} \min_{p \in Paths(s,v)} \text{ capacity}(p) \end{pmatrix} \right) \end{pmatrix}$$

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:

$$LT_{RUST}(s) = \text{let } radius \coloneqq \begin{pmatrix} \text{ilet } \langle x, y \rangle \coloneqq \mathcal{R}' \ \mathcal{F} \text{ in} \\ \text{mlet } \langle x', y' \rangle \coloneqq \langle x, y \rangle \text{ in} \\ \text{rlet } \langle x'', y'' \rangle \coloneqq \mathcal{R}'' \langle x', y' \rangle \text{ in} \\ \min(x'', y'') \end{pmatrix} \text{ in} \\ \mathcal{R}_{v \in V} \left(\text{ilet } x \coloneqq \min_{s} \text{ weight in } x < radius, \text{ ilet } y \coloneqq \min_{s} \text{ capacity in } y \right) \end{cases}$$

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

$$LT_{RUST}(s) = \text{let } radius \coloneqq \begin{pmatrix} \text{ilet } \langle x, y \rangle \coloneqq \mathcal{R}' \mathcal{F} \text{ in} \\ \text{mlet } \langle x', y' \rangle \coloneqq \langle x, y \rangle \text{ in} \\ \text{rlet } \langle x'', y'' \rangle \coloneqq \mathcal{R}'' \langle x', y' \rangle \text{ in} \\ \min(x'', y'') \end{pmatrix} \text{ in} \\ \mathcal{R}_{v \in V} \quad \begin{pmatrix} \text{ilet } \langle x, y \rangle \coloneqq \langle \min_{s} \text{ weight, } \min_{s} \text{ capacity} \rangle \text{ in } \langle x < radius, y \rangle \end{pmatrix}$$

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

$$LTRUST(s) = \operatorname{let} radius \coloneqq \begin{pmatrix} \operatorname{ilet} \langle x, y \rangle \coloneqq \mathcal{R}' \quad \mathcal{F} \text{ in} \\ \operatorname{mlet} \langle x', y' \rangle \coloneqq \langle x, y \rangle \operatorname{in} \\ \operatorname{rlet} \langle x'', y'' \rangle \coloneqq \mathcal{R}'' \langle x', y' \rangle \operatorname{in} \\ \operatorname{min}(x'', y'') \\ \mathcal{R}_{v \in V} \left(\operatorname{ilet} \langle x, y \rangle \coloneqq \mathcal{R}''' \quad \mathcal{F}' \operatorname{in} \langle x < radius, y \rangle \right) \\ \operatorname{where} \quad \mathcal{F}' \coloneqq \lambda p. \quad \langle \operatorname{weight}(p), \operatorname{capacity}'(p) \rangle \\ \mathcal{R}'''(\langle a, b \rangle, \langle a', b' \rangle) \coloneqq \\ \langle \operatorname{min}(a, a'), \operatorname{min}(b, b') \rangle \\ \end{cases}$$

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

$$LT_{RUST}(s) = \text{let } radius \coloneqq \begin{pmatrix} \text{ilet } \langle x, y \rangle \coloneqq \mathcal{R}' & \mathcal{F} \text{ in} \\ \text{mlet } \langle x', y' \rangle \coloneqq \langle x, y \rangle \text{ in} \\ \text{rlet } \langle x'', y'' \rangle \coloneqq \mathcal{R}'' & \langle x', y' \rangle \text{ in} \\ \min(x'', y'') \\ \begin{pmatrix} \text{ilet } \langle x, y \rangle \coloneqq \mathcal{R}'' & \mathcal{F}' \\ \text{mlet } \langle x', y' \rangle \coloneqq \min(x < radius, y) \\ \text{rlet } x'' \coloneqq \mathcal{R} & \langle x', y' \rangle \text{ in} \\ x'' \end{pmatrix}$$

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

1030 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. DS(s) $\bigcup_{v \in V \land \left(\min_{p \in Paths(s,v)} weight(p)\right) > 7}$ = By VSel $= \mathcal{R}_{v \in V} \left\langle \left(\min_{p \in \mathsf{Paths}(s,v)} \mathsf{weight}(p) \right) > 7, \{v\} \right\rangle \text{ where } \begin{array}{c} \mathcal{R}(\langle a, b \rangle, \langle a', b' \rangle) \coloneqq \\ \text{if } (a \land a') \text{ then } \langle a, b \cup b' \rangle \\ \text{else}(a) \text{ then } \langle a, b \rangle \end{array}$ By FPRED and FILIT $= \Re_{x \in V} \left\langle \left(\text{ilet } x \coloneqq \min_{s} \text{ weight in } x \right) > \text{ilet } x' \coloneqq \bot \text{ in 7,} \right\rangle$ $\mathcal{R}(\langle a, b \rangle, \langle a', b' \rangle) \coloneqq$ if $(a \land a')$ then $\langle a, b \cup b' \rangle$ ilet $x'' \coloneqq \perp \text{ in } \{v\}$ where By FILETBIN else(*a*) then $\langle a, b \rangle$ else $\langle a', b' \rangle$ $= \mathcal{R}_{v \in \mathsf{V}} \left(\text{ilet } \langle \langle x, x' \rangle, x'' \rangle \coloneqq \langle \langle \min_{s} \text{ weight, } \bot \rangle, \bot \rangle \text{ in } \langle x > 7, \{v\} \rangle \right)$ By FMPAIR' $= \mathcal{R}_{v \in V} \left(\text{ilet } x := \min_{s} \text{ weight in } \langle x > 7, \{v\} \rangle \right)$ By FVRed else $\langle a', b' \rangle$

1079				
1080	DRR	_	DIAM	
1081	Diat	_	Radius	
1082			max max min length(a)	
1083		_	$s \in \{s_1, s_2\}$ $v \in V$ $p \in Paths(s, v)$	Cimilante Fig. 9 fee
1084		=	$\min_{p} \max_{p} \min_{p} ength(p) $	Similar to Fig. 2 for
1085			$s \in \{s_1, s_2\} \ v \in V \ p \in Paths(s, v)$	PADUE in the never
1086				RADIUS III the paper.
1087			(ilet $\langle x_1, y_1 \rangle := \min \mathcal{F}$ in)	
1088			$\langle s_1, s_2 \rangle$	
1089			mlet $\langle x'_1, y'_1 \rangle \coloneqq \langle x_1, y_1 \rangle$ in	
1090			$\mathcal{F} \coloneqq \lambda p. \langle \operatorname{length}(p), \operatorname{length}(p) \rangle$	
1091		=	$\frac{(\max(x_1, y_1))}{(\max(x_1, y_1))} \text{where} \mathcal{R}(\langle a, b \rangle, \langle a', b' \rangle) \coloneqq$	By FLETSBIN
1092			$(\text{ilet } \langle x_2, y_2 \rangle \coloneqq \min_{\langle s_1, s_2 \rangle} \mathcal{F} \text{ in } \\ \langle \max(a, a'), \max(b, b') \rangle$	
1093			mlet $\langle x'_2, y'_2 \rangle \coloneqq \langle x_2, y_2 \rangle$ in	
1094			rlet $\langle x_2^{\prime\prime}, y_2^{\prime\prime} \rangle \coloneqq \mathcal{R} \langle x_2^{\prime}, y_2^{\prime} \rangle$ in	
1095			$\min(x_2^{\prime\prime}, y_2^{\prime\prime}) \qquad \qquad$	
1096				
1097			(ilet $\langle \langle x_1, y_1 \rangle, \langle x_2, y_2 \rangle \rangle \coloneqq \langle \min_{\langle x_1, y_2 \rangle} \mathcal{F}, \min_{\langle x_2, y_2 \rangle} \mathcal{F} \rangle$ in	By IELIM
1098			$ (x_1, x_2) (x_1, x_2, y_2) = \langle \langle x_1, y_1 \rangle, \langle x_2, y_2 \rangle \rangle $ in	Common
1099		=	rlet $\langle \langle x_1', y_1' \rangle, \langle x_2', y_2' \rangle \rangle \coloneqq \langle \mathcal{R} \langle x_1', y_1' \rangle, \mathcal{R} \langle x_2', y_2' \rangle \rangle$ in	reduction
1100			$\left(\frac{1}{\max(x_1'', y_1'')} / \frac{1}{\min(x_2'', y_2'')} \right)$	elimination
1101				chilination
1102				
1103			(ilet $\langle x_1, y_1 \rangle := \min \mathcal{F}$ in ()	Similarly,
1104			$\langle s_1, s_2 \rangle$	by Common
1105		=	$ \text{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 \text{ in } $	vertex-based
1106			$ \max(x'', y'', x', y'', y'') = \langle \mathbf{x} (x_1, y_1), \mathbf{x} (x_2, y_2) \rangle $ max(x'', y'')/min(x'', y'')	mapping
1107			$(\max(x_1, g_1) / \min(x_2, g_2))$	elimination
1108				
1109			$(: I_{i} \downarrow_{i} $	Similarly
1110			$(\operatorname{Ilet} \langle x_1, y_1 \rangle := \min_{\langle s_1, s_2 \rangle} \mathcal{F} \operatorname{In}$	by Common
1111		=	mlet $\langle x'_1, y'_1 \rangle \coloneqq \langle x_1, y_1 \rangle$ in	vertex-based
1112			$\operatorname{rlet} \langle \langle x_1'', y_1'' \rangle, \langle x_2'', y_2'' \rangle \rangle \coloneqq \langle \mathcal{R} \langle x_1', y_1' \rangle, \mathcal{R} \langle x_1', y_1' \rangle \rangle \text{ in }$	reduction
1113			$\max(x_1'', y_1'') / \min(x_2'', y_2'')$	elimination
1114				
1115				
1116			$\left(\begin{array}{c} \text{ilet } \langle x_1, y_1 \rangle \coloneqq \min_{\langle s_1, s_2 \rangle} \mathcal{F} \text{ in} \\ \mathcal{F} \coloneqq \lambda p, \langle \text{length}(p), \text{length}(p) \rangle \right)$	
111/		=	mlet $\langle x'_1, y'_1 \rangle := \langle x_1, y_1 \rangle$ in where $\mathcal{R}(\langle a, b \rangle, \langle a', b' \rangle) :=$	
1118			$ \operatorname{rlet} \langle x_1'', y_1'' \rangle \coloneqq \mathcal{R} \langle x_1', y_1' \rangle \text{ in } \langle \max(a, a'), \max(b, b') \rangle$	
1119			$\max(x_1'', y_1'') / \min(x_1'', y_2'')$	
1120				
1121				
1122				
1123				
1124				
1126				
1120				

1128 3 Mapping Specification to Iteration-Map-Reduce

¹¹²⁹ 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.

1134 3.1.1 Pull Model

Pull model with idempotent reduction.

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

The full proof is available in the appendix § 4.4.1. We prove by induction that after each iteration k, the value $S_{\text{pull}+}^{k}(v)$ of each vertex v is $Spec^{k}(v)$ that is the reduction over paths to v of length less than k. At the iteration k = 1, the specification $Spec^{1}(v)$ 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 Iproperly initializes each vertex v to $Spec^{1}(v)$. 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 $l, 0 \le l < k$. 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 $l, 0 \le l < k$. 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 $l, 0 \le l < k + 1$. 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.

THEOREM 9 (CORRECTNESS OF PULL (NON-IDEMPOTENT REDUCTION)). For all $\mathcal{R}, \mathcal{F}, \mathcal{I}, \mathcal{P}, k \ge 1$, and s, let C(p) := (head(p) = s), if the conditions $\mathbb{C}_1 - \mathbb{C}_8$ hold, and s is not on any cycle, $S_{\text{pull}-}^k(v) =$ $Spec^k(v)$.

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, $0 \le l < k$. 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.

1226 3.1.2 Push Model

1227 Push model with idempotent reduction.

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

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 vthat 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} , I, \mathcal{P} , $k \ge 1$, and s, let C(p) := (head(p) = s), if the conditions $\mathbb{C}_1 - \mathbb{C}_8$ hold, and s is not on any cycle, $S^k_{\text{push}-}(v) = Spec^k(v)$

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 k1324 be represented as $S^k_{\text{push}-}(v)$. The main difference with the previous model is that every changed 1325 predecessor u_i first rollbacks its previous update before applying its new update. The rollback 1326 1327 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 1328 that is propagated to v to be rolled back. The rollback value is expected to cancel the previously 1329 propagated value. For example, for the PAGERANK use-case as Fig. 7 shows, the rollback function 1330 returns the negation of the previously propagated value. For each predecessor u_i , the rollback 1331 function \mathcal{B} is applied to the previous value $\mathcal{S}_{\text{push}-}^{k-1}(u_i)$ of u_i and the edge $\langle u_i, v \rangle$, and the propagate 1332 function \mathcal{P} is applied to the latest value $\mathcal{S}_{\text{push-}}^{k'}(u_i)$ of u_i and the edge $\langle u_i, v \rangle$. The two resulting 1333 values are reduced with the current value of v. 1334

1335 Definition 7 (Push (non-idempotent reduction) II).

1336 1337

1338 1339

1340

1341 1342

1347

1352

1353

1354

1355

1356

1357

1358

1359

1360

1361

1364

1365

1366

 $S^{0}_{push-}(v) \coloneqq \bot$ $S^{1}_{push-}(v) \coloneqq I(v)$ $S^{k+1}_{push-}(v) \coloneqq \mathcal{E}(S_{n}), \quad k \ge 1 \quad where$ $let \{u_{0}, ..., u_{n-1}\} \coloneqq CPreds^{k}(v) \text{ in}$ $S_{0} \coloneqq S^{k}_{push-}(v)$ $S_{i+1} \coloneqq \mathcal{R}(\mathcal{R}(S_{i}, \mathcal{B}\left(\mathcal{S}_{push-}^{k-1}(u_{i}), \langle u_{i}, v \rangle\right)),$ $\mathcal{P}\left(S^{k}_{push-}(u_{i}), \langle u_{i}, v \rangle\right))$

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

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

As we saw in Def. 7, in this variant of push model with non-idempotent reduction $S_{push-}^k(v)$, 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 $Spec^k(v)$.

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

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 \perp 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 $l, 0 \le l < k + 1$ that the specification $Spec^{k+1}(v)$ requires.

NUMBER OF SHORTEST PATHS (NSP) $I := \lambda v.$ if $(v = s) \langle 0, 1 \rangle$ else \perp

 $\mathcal{P} := \lambda n. \text{ if } (b - s) \langle 0, 1 \rangle \text{ else } \perp$ $\mathcal{P} := \lambda n, e. n + \text{weight}(e)$ $\mathcal{R} := \lambda \langle w, n \rangle, \langle w', n' \rangle.$ $\text{ if } (w = w') \langle w, n + n' \rangle$ $\text{ else if } (w > w') \langle w', n' \rangle$ $\text{ else } \langle w, n \rangle$ $\mathcal{E} := \lambda n. n$ $\mathcal{B} := \lambda \langle w, n \rangle, e. \langle w, -n \rangle$

Fig. 16. The number of shortest paths

Anon.

Asynchronous Model 3.1.3 The predecessors of the vertex *v* that changed value in the iteration *k*: $CPreds^{k}(v) = \left\{ u \mid u \in preds(v) \land S^{k}(u) \neq S^{k-1}(u) \right\}$ **DEFINITION 8 (PULL (IDEMPOTENT REDUCTION)).** $\begin{aligned} \mathcal{S}^{0}_{\text{apull+}}(v) &\coloneqq \bot \\ \mathcal{S}^{1}_{\text{apull+}}(v) &\coloneqq I(v) \end{aligned}$ $S_{\text{apull+}}^{k+1}(v) \coloneqq \begin{cases} S_{\text{apull+}}^{k}(v) & \text{if CFr} \\ \mathcal{E}\left[\mathcal{R}\left(S_{\text{apull+}}^{k}(v), \mathcal{R}_{u \in \text{preds}(v)} \mathcal{P}\left(S_{\text{apull+}}^{k}(u), \mathcal{S}_{\text{apull+}}^{k+1}(u), \langle u, v \rangle\right)\right)\right] & \text{else} \end{cases}$ $\textit{if } \mathsf{CPreds}^k(v) = \emptyset$ $k \ge 1$ DEFINITION 9 (PULL (NON-IDEMPOTENT REDUCTION)).
$$\begin{split} \mathcal{S}^{0}_{\text{apull}-}(v) &\coloneqq \bot \\ \mathcal{S}^{1}_{\text{apull}-}(v) &\coloneqq \mathcal{I}(v) \end{split}$$
$$\begin{split} \mathcal{S}_{\mathsf{apull}-}^{k+1}(v) \coloneqq \begin{cases} \mathcal{S}_{\mathsf{apull}-}^k(v) & \text{if } \mathsf{CPreds}^k(v) = \emptyset \\ \mathcal{E}\left[\mathcal{R}_{u \in \mathsf{preds}(v)} \, \mathcal{P}\!\left(\mathcal{S}_{\mathsf{apull}-}^k(u) \, ? \, \mathcal{S}_{\mathsf{apull}-}^{k+1}(u), \langle u, v \rangle \right) \right] & \text{else} \end{cases} \end{split}$$
 $k \ge 1$ DEFINITION 10 (PUSH (IDEMPOTENT REDUCTION)).
$$\begin{split} \mathcal{S}^{0}_{&\text{apush+}}(v) &\coloneqq \bot \\ \mathcal{S}^{1}_{&\text{apush+}}(v) &\coloneqq \mathcal{I}(v) \\ \mathcal{S}^{k+1}_{&\text{apush+}}(v) &\coloneqq \mathcal{E}(S_{n}), \quad k \geq 1 \quad \text{where} \end{split}$$
let $\{u_0, ..., u_{n-1}\} \coloneqq CPreds^k(v)$ in let $m_i := |CPreds^k(u_i)|$ in $S_0(v) \coloneqq S_{\text{apush}+}^k(v)$ $S_{i+1}(v) \coloneqq \mathcal{R}\left(S_i(v), \mathcal{P}\left(\sum_{j \in \{1, \dots, m_i\}} S_j(u_i), \langle u_i, v \rangle\right)\right)$ DEFINITION 11 (PUSH (NON-IDEMPOTENT REDUCTION)). $\begin{aligned} &\mathcal{S}^{0}_{\mathsf{apush-}}(v)\coloneqq\bot\\ &\mathcal{S}^{1}_{\mathsf{apush-}}(v)\coloneqq I(v) \end{aligned}$ $\mathcal{S}_{apush-}^{k+1}(v) \coloneqq \mathcal{E}(S_n(v)), \quad k \ge 1 \quad where$ let $\{u_0, ..., u_{n-1}\} \coloneqq \mathsf{CPreds}^k(v)$ in $S_0(v) \coloneqq \mathcal{S}_{apush-}^k(v)$ $S_{i+1}(v) \coloneqq \mathcal{R}(\mathcal{R}(S_i(v), v))$ $\mathcal{B}\left(b^{k-1}(u_i), \langle u_i, v \rangle\right),$ $\mathcal{P}\left(b^k(u_i), \langle u_i, v \rangle\right),$ $b^{k+1}(v) \coloneqq ?_{i \in \{0..n\}} S_i(v)$ 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 increase that were precented in Fig. 6 are synchronous in the synchronous increase increase in the synchronous in the synchronous in the synchronous increase increase in the synchronous in the synchronous in the synchronous increase in the synchronous in the sync

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 $S^k(v)$, the old value $S^{k+1}(v)$ 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 $S_{apull+}^{k}(u)$ or the new value $S_{apull+}^{k+1}(u)$ 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 $S_{push+}^{k}(u_i)$ 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.

1486 The asynchronous push model for non-idempotent reduction functions is presented in Def. 11. 1487 It not similar to the corresponding synchronous definition presented in Def. 4. The difference is 1488 that the values propagated by a vertex can be any of its intermediate values and not necessarily 1489 its value at the end of the last iteration. Thus, we need to store the previously propagated values 1490 to roll them back before propagating new values. Consider a vertex v and its predecessor u_i . The 1491 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 1492 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 1493 new value $b^k(u_i)$ is propagated by the propagation function \mathcal{P} . 1494

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

1496 DEFINITION 12 (PATHS).
$$P^{\infty}(v) = \{p \mid p \in \text{Paths}(v) \land C(p)\}$$

1495

1497

1498

1499

1501

1502

1505

1506 1507

1508 1509

1510

1511

1512

1513 1514

1515

1516

1517

1518 1519 The definition of specification Spec(v) is the same as definition Def. 5; only the paths are factored to $P^{\infty}(v)$.

1500 DEFINITION 13 (SPECIFICATION). $Spec(v) = \mathcal{R}_{p \in P^{\infty}(v)} \mathcal{F}(p)$

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

1503 DEFINITION 14 (k-PATHS). $P^k(v) = \{p \mid p \in \text{Paths}(v) \land C(p) \land \text{length}(p) < k\}$ 1504

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

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

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, $aSpec^{k}(v)$, as set of values: the reductions of any set of paths P in $aP^{k}(v)$.

Definition 17 (A-k-Specification). $aSpec^{k}(v) = \{\mathcal{R}_{p \in P} \mathcal{F}(p) | P \in aP^{k}(v)\}$

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 $aSpec^{k}(v)$.

THEOREM 13 (CORRECTNESS OF PULL (IDEMPOTENT REDUCTION)). Forall $\mathcal{R}, \mathcal{F}, C, I, \mathcal{P}, and k \geq 1$, if the conditions $\mathbb{C}_1 - \mathbb{C}_4$ and $\mathbb{C}_6 - \mathbb{C}_9$ hold, then $S^k_{apull+}(v) \in aSpec^k(v)$

The proof is similar to the proof of Theorem 8. The set of paths covered by $S_{\text{apull}+}^{k+1}(u)$ is a superset of path covered by $S_{\text{apull}+}^{k}(u)$. 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 $\mathcal{R}, \mathcal{F}, I, \mathcal{P}, k \ge 1$, and s, let C(p) := (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, $S_{\text{apull}}^k(v) \in aSpec^k(v)$

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

THEOREM 15 (CORRECTNESS OF PUSH (IDEMPOTENT REDUCTION)). For all $\mathcal{R}, \mathcal{F}, C, I, \mathcal{P}, and k \geq 1$, if the conditions $\mathbb{C}_1 - \mathbb{C}_4$ and $\mathbb{C}_6 - \mathbb{C}_9$ hold, $S^k_{apush+}(v) \in aSpec^k(v)$

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 $S_{\text{push}+}^k(u_i)$. 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 $\mathcal{R}, \mathcal{F}, C, I, \mathcal{P}, and k \geq 1$, if the conditions $\mathbb{C}_1 - \mathbb{C}_8$ hold, $S^k_{apush-}(v) \in aSpec^k(v)$

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 $S_{pull-}^k(u_i)$. 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 $k \ge k'$, $aSpec^k(v) = \{Spec(v)\}$.

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, *aSpec*^k(v) = {*Spec*(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, $aSpec^k(v) = \{\mathcal{R}_{p \in P^{l+1}(v)} \mathcal{F}(p)\}$. Similarly, it can be shown that $Spec(v) = \{\mathcal{R}_{p \in P^{l+1}(v)} \mathcal{F}(p)\}$. Therefore, $aSpec^k(v) = \{Spec(v)\}$.

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 $\mathcal{R}, \mathcal{F}, C, I$, 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 $S^k_{apull+}(v) = Spec(v)$

1522

1523

1524 1525

1526

1527

1528 1529

1530

1531 1532

1536

1537

1538 1539

1540

1541

1542 1543

1544 1545

1546

1547

1548 1549

1550

1551

1563

1564

1565

3.1.4 Streaming Graphs 1569

1584

1602

1603

1604

In contrast to a static graph, a streaming graph can continuously change in response to external 1570 events. Thus, to have up-to-date results, the graph analytics computations should be periodically 1571 repeated. Stream graph processing strives to benefit from the results computed prior to the updates 1572 instead of restarting the iteration form the initial values. The idea is that starting from the prior result 1573 can accelerate the convergence. What are the conditions such that the incremental computation 1574 yields the correct results? We first consider addition and then removal of edges and present the 1575 1576 correctness conditions for incremental commutation after each.

Incremental Computation. Consider a graph G. Let us denote the result of a path-based 1577 reduction Spec(v) on G as $Spec_G(v)$. Let $G + \delta$ be the result of updating (adding or removing) an 1578 edge $e = \langle s_e, t_e \rangle$ in *G*. The incremental computation on $G + \delta$ starts from the prior result $Spec_G(v)$ 1579 for G. The incremental pull model is similar to the basic model (of Def. 1). The difference is that 1580 1581 (1) the starting state is $Spec_G(v)$ instead of 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 1582 incremental computation at iteration k denoted as $S_{G+\delta}^k(v)$ is defined as follows: 1583

DEFINITION 18 (INCREMENTAL PULL MODEL (WITH IDEMPOTENT REDUCTION)).

Addition of Edges. If the update δ in $G + \delta$ is adding an edge, does the result of incremental 1591 computation $S_{G+\delta}^k(v)$ converge to its specification Spec(v)? It turns out that it does with the same 1592 conditions as the static case. Adding an edge only increases the set of paths. The prior value of 1593 a vertex is the result of reduction on the old set of paths to that vertex. That set may now be 1594 incomplete. However, the prior values can help the incremental computation skip most of the 1595 initial iterations. For example, in the shortest path SSSP use-case, the newly added edge may 1596 improve the previously found shortest path only for some of the vertices. Subsequent iterations 1597 will eventually reduce the values of all the new paths with the prior values of the vertices. As the 1598 reduction function is assumed to be commutative, associative and idempotent, the reduction order 1599 and repeated reductions of a path do not affect the result. Thus, we can state the following theorem 1600 for the correctness of incremental reduction after adding edges. 1601

THEOREM 19 (CORRECTNESS AFTER ADDING EDGES). For all $\mathcal{R}, \mathcal{F}, I$ 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 $S_{G+\delta}^k(v) = Spec(v)$.

Removal of Edges. In contrast to adding, if the update is removing an edge, the incremental 1605 computation is not necessarily correct. When an edge $\langle s_e, t_e \rangle$ is removed, the value of t_e becomes 1606 incorrect if it has been calculated using the value of s_e . Thus, the incremental computation (Def. 18) 1607 1608 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 1609 back to one of its predecessors u, and the value of u has been calculated based on the old value of 1610 t_e , the recalculated value of t_e is still incorrect. The new value of t_e can lead to calculation of new 1611 values back to u and then again for t_e . The question is whether the iterative calculation around the 1612 loop eventually forgets the incorrect value. It turns out that it does, if extending a path with an 1613 edge makes the value of the path less favorable during reduction. For example, in the SSSP use-case, 1614 the value of a path is its weight and the weight of an extended path increases; thus, the extended 1615 path is less favorable for the min reduction function. The cycle can take only larger values back to 1616 1617

Anon.



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.

1634 However, in the CC use-case, the value of 1635 a path is the identifier of its source; thus, the 1636 value of an extended path stays the same. Con-1637 sider the graph in Fig. 18b where two cycles 1638 are connected by the edge $e = \langle s_e, t_e \rangle$ where 1639 the cycle on the s_e side has the vertex with the

$$\begin{aligned} & \text{Streaming:} \\ & \mathbb{C}_{12} \text{ (Worsening):} \\ & \forall p, e. \ \mathcal{R}(\mathcal{F}(p), \mathcal{F}(p \cdot e)) = \mathcal{F}(p) \neq \mathcal{F}(p \cdot e) \end{aligned}$$

Fig. 19. Correctness and Termination Conditions

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.

THEOREM 20 (CORRECTNESS AFTER REMOVING EDGES). For all \mathcal{R} , \mathcal{F} , 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 $S^k_{G+\delta}(v) = Spec(v)$.

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.

32

1628

1629 1630

1647

1648

1649

1650

1651

1652

1653

1654

1655

1667 3.1.5 Factored Path-based Reductions

¹⁶⁶⁸ Consider the factored path-based reduction \mathcal{RF} with a general configuration *c*. We show that the ¹⁶⁶⁹ correctness conditions for its iterative execution are captured by the conditions that were presented ¹⁶⁷⁰ in Fig. 13.

Let us define C^c as follows:

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

¹⁶⁷⁴ Let us define \mathcal{F}^c as follows:

1678 The specification of the factored path-based reduction $\mathcal{R} \mathcal{F}$ is the following:

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

that can be captured by the specification Spec(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 $\mathcal{R} \mathcal{F}$ 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

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

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

are the following

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

$$\tag{4}$$

¹⁶⁹⁴ This means that the initialization for each element of the state tuple mirrors the initialization ¹⁶⁹⁵ conditions \mathbb{C}_1 and \mathbb{C}_2 .

Synthesis of Iterative Reduction 1716 3.2

1717 To find candidate expressions for the body of the kernel functions, we apply a type-guided enumer-1718 ative search to the expression grammar presented in Fig. 20b. The expression constructors have 1719 union types; for example, the plus operator + can be applied to both integers Int and floating point 1720 Float numbers. The procedure Candidates in Fig. 20a returns the set of expressions of the input type 1721 T and size size. It is a recursive procedure that uses memoization to avoid redundant enumeration. 1722 It keeps a map from types to maps from sizes to the set of previously synthesized expressions. To 1723 synthesize an expression of type T, it only considers the expression constructors with the return 1724 type T. A constructor itself uses one unit of size. For each constructor c, the Candidates procedure 1725 considers all the possible distributions of the remained size, that is size - 1, between the parameters 1726 of c. For each distribution, it recursively obtains a set of expressions E_i for each parameter p_i using 1727 its type and its allocated size. It then applies c to each element of the product of the sets E_i to yield 1728 candidate expressions. It memoizes and returns the set of these candidates.

1729 The functions Fig. 20c and Fig. 20d synthesize the functions I and \mathcal{P} . We consider synthesis for 1730 \mathcal{P} ; synthesis for *I* is similar. Fig. 20d presents the Synth \mathcal{P} procedure that given the path function 1731 $\mathcal F$ and the reduction function $\mathcal R$ of a path-based reduction, synthesizes the propagation function $\mathcal P$. 1732 It starts by memoizing expressions of size one, variables and literals, to make them available for 1733 the synthesis of the body of \mathcal{P} . Let *T* be the return type of \mathcal{F} ; vertices store values of type *T*. The 1734 propagation function \mathcal{P} takes a value stored at a vertex (of type of *T*) and an edge (of type Edge) 1735 and returns a vertex value (of type T). Thus, the two input variables of the two input types, the 1736 variable n of type T and the variable l of type Edge, are memoized as available expressions. Then, 1737 candidate expressions of type T are obtained from the Candidates procedure. Expressions of larger 1738 sizes are incrementally checked as candidate bodies for \mathcal{P} .

1739 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} 1740 is replaced by the candidate. We use the notation of $\mathcal{A} \vdash \mathcal{A}'$ to represent whether the assertion 1741 \mathcal{A}' is valid in the context of the assumed assertion(s) \mathcal{A} . To check the validity of an assertion, we 1742 use off-the-shelf SMT solvers to check the satisfiability of its negation. The context of the validity 1743 check $\mathcal{F}; \mathcal{R}; \Gamma$ is the definition of the functions \mathcal{F} and \mathcal{R} from the given path-based reduction, and 1744 a set of assertions Γ that define basic graph functions and relations.

1745 Fig. 21 represents the context assertions Γ : assertions for the path functions length, weight, 1746 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 $\langle V, V \rangle$ 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 1751 the weight of the path without the last edge and the edge weight of the last edge. 1752

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 .

34

- 1747 1748 1749 1750
- 1753 1754
- 1755 1756 1757
| 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 $size$ | e
n
v
T | $ \begin{array}{rrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrr$ |
|--|---|---|
| return <i>E</i>
(a) Type-guided expression
enumeration | | (b) Grammar |
| def Synth $I (\mathcal{F})$
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 $E \leftarrow Candidates(return type of \mathcal{F}, size)I_7 foreach (e \in E)I_8 if \mathcal{F}; \Gamma \vdash (\mathbb{C}_1 \land \mathbb{C}_2)[I \mapsto (\lambda v. e)]I_9 return (\lambda v. e)I_{10} size \leftarrow size + 1(c) Synthesis of the initialization function I$ | def
P ₁
P ₂
P ₃
P ₄
P ₅
P ₆
P ₇
P ₈
P ₉
P ₁₀
P ₁₁ | f Synth $\mathcal{P}(\mathcal{F}, \mathcal{R})$
let <i>T</i> be the return type of \mathcal{F} .
memoize variable <i>n</i> for <i>T</i> and size 1
memoize variable <i>l</i> for type Edge and size 1
foreach (literal l_i with type T_i)
memoize l_i for T_i and size 1
size $\leftarrow 1$
while (true)
$E \leftarrow \text{Candidates}(T, size)$
foreach $(e \in E)$
if $\mathcal{F}; \mathcal{R}; \Gamma \vdash (\mathbb{C}_4 \land \mathbb{C}_5)[\mathcal{P} \coloneqq (\lambda n, l. e)]$
return $(\lambda n, l. e)$
1 size \leftarrow size + 1
(d) Synthesis of the propagation function \mathcal{P} |
| Fig. 20. Synthesis Grammar | r and | l functions |

 I_1

 I_{10}

	$P \coloneqq List[V],$	
		weight: $\langle V,V\rangle \to \mathbb{N}$
	length: $P \rightarrow \mathbb{N}$	$\forall v. \text{ eweight}(\langle v, v \rangle) = 0$
	elength: $\langle V, V \rangle \rightarrow \mathbb{N}$	$\forall p. \text{ if } (p = \bot) \text{ weight}(p) = 0$
	$\forall \langle u, v \rangle$. If $(u = v)$ elength $(\langle u, v \rangle) = 0$	else $ x + y = x + y + x + y + x + y $
	else $(1 + 1)$ $(1 + 1)$	let $v := nead(p), p^{-} := tall(p) ln$
	elength $\langle u, v \rangle = 1$	$\prod_{p \in P} (p = 1) \text{weight}(p) = 0$
	p . If $(p = \pm)$ length $(p) = 0$	else $let v' := head(v')$ in
	let $v := head(v)$, $v' := tail(v)$ in	weight(p) –
	if $(p' = 1)$ length $(p) = 0$	weight(p') + eweight($\langle n' n \rangle$)
	else	
$\Gamma =$	let $v' := head(p')$ in	capacity: $P \rightarrow \mathbb{N}$
	length(p) =	ecapacity: $\langle V, V \rangle \rightarrow \mathbb{N}$
	length(p') + elength($\langle v', v \rangle$)	$\forall v. \text{ ecapacity}(\langle v, v \rangle) = \bot$
		$\forall p. \text{ if } (p = \bot) \text{ capacity}(p) = \bot$
	penultimate: $P \rightarrow V$	else
	$\forall p. \text{ if } (p = \bot) \text{ penultimate}(p) = \bot$	let $v \coloneqq head(p), p' \coloneqq tail(p)$ in
	else	if $(p' = \bot)$ capacity $(p) = \bot$
	let $v \coloneqq head(p), p' \coloneqq tail(p)$ in	else
	if $(p' = \bot)$ penultimate $(p) = v$	let $v' \coloneqq head(p')$ in
	else	capacity(p) =
	penultimate(p) = head(p')	$\min(\operatorname{capacity}(p'),\operatorname{ccapacity}(\langle v',v\rangle))$
	5. 24. 0. 1	
	Fig. 21. Conte	xt assertions I
	Γ=	$P := \text{List}[V],$ $\operatorname{length}: P \to \mathbb{N}$ $\operatorname{elength}: \langle V, V \rangle \to \mathbb{N}$ $\forall \langle u, v \rangle. \text{ if } (u = v) \operatorname{elength}(\langle u, v \rangle) = 0$ else $\operatorname{elength}(\langle u, v \rangle) = 1$ $\forall p. \text{ if } (p = \bot) \operatorname{length}(p) = 0$ else $\operatorname{let} v := \operatorname{head}(p), p' := \operatorname{tail}(p) \text{ in }$ $\operatorname{length}(p) = 0$ else $\operatorname{let} v' := \operatorname{head}(p') \text{ in }$ $\operatorname{length}(p') = 0$ else $\operatorname{let} v := \operatorname{head}(p') + \operatorname{elength}(\langle v', v \rangle)$ $Vp. \text{ if } (p = \bot) \text{ penultimate}(p) = \bot$ else $\operatorname{let} v := \operatorname{head}(p), p' := \operatorname{tail}(p) \text{ in }$ $\operatorname{if } (p' = \bot) \text{ penultimate}(p) = v$ else $\operatorname{penultimate}(p) = \operatorname{head}(p')$ $Fig. 21. Conte$

1863	4 Proofs
1864	4.1 Helper Definitions
1865	DEFINITION 19 (SUBSTITUTION).
1866	Substitution $E := N$:
180/	$N \coloneqq n \mid \langle N, N \rangle$
1860	
1870	$\langle E, E' \rangle [X \coloneqq N] = \langle E[X \coloneqq N], E'[X \coloneqq N] \rangle$
1871	$e[\langle X, X' \rangle := \langle N, N' \rangle] = e[X := N][X' := N']$
1872	$e \oplus e'[x \coloneqq n] = e[x \coloneqq n] \oplus e'[x \coloneqq n]$
1873	$x[x \coloneqq n] = n$
1874	$x [x := n] = x^n$
1875	The definitions of substitution for $a := D E := D$ and $P := D$ are similar
1876	The definitions of substitution for $e := D, L := D$, and $K := D$ are similar. $D := d \mid (D \mid D)$
1877	$D := u \mid \langle D, D \rangle$
1878	
1879	
1880	
1881	
1882	
1883	
1884	
1886	
1887	
1888	
1889	
1890	
1891	
1892	
1893	
1894	
1895	
1896	
1897	
1898	
1899	
1900	
1902	
1903	
1904	
1905	
1906	
1907	
1908	
1909	
1910	
1911	

```
1912
         4.2 Semantics Compositionality
1913
            LEMMA 2 (COMPOSITIONALITY FOR r).
1914
             For all r, r' and \mathbb{R}, if \llbracket r \rrbracket = \llbracket r' \rrbracket then \llbracket \mathbb{R}[r] \rrbracket = \llbracket \mathbb{R}[r'] \rrbracket.
1915
         Proof.
1916
         Induction on \mathbb{R}:
1917
              Case
1918
                   (1) \mathbb{R} = []
1919
                   Immediate.
1920
              Case
1921
                   (2) \mathbb{R} = \mathbb{R}' \oplus r
1922
                   Immediate by the rule SRBIN.
1923
1924
1925
            LEMMA 3 (COMPOSITIONALITY FOR m).
1926
             For all m, m', and M, if [m] = [m'] then [M[m]] = [M[m']].
1927
         Proof.
1928
         Induction on \mathbb{M}:
1929
              Case
1930
                   (1) M = []
1931
                   Immediate.
1932
               Case
1933
                   (2) \mathbb{M} = \mathcal{R} \mathbb{M}'
1934
1935
                   Immediate by the rule SVRED.
              Case
1936
                   (3) \mathbb{M} = \mathbb{M}' \oplus m
1937
1938
                   Immediate by the rule SMBIN.
1939
               Case
1940
                   (4) \mathbb{M} = m \oplus \mathbb{M}'
                   Immediate by the rule SMBIN.
1941
1942
1943
            LEMMA 4 (COMPOSITIONALITY FOR M).
1944
             For all M, M', and \mathbb{M}s, if \llbracket M \rrbracket = \llbracket M' \rrbracket then \llbracket \mathbb{M}s[M] \rrbracket = \llbracket \mathbb{M}s[M'] \rrbracket.
1945
         Proof.
1946
         Induction on \mathbb{M}s:
1947
              Case
1948
                   (1) Ms = []
1949
                   Immediate.
1950
1951
              Case
                   (2) \mathbb{M}\mathbf{s} = \langle \mathbb{M}\mathbf{s}, M \rangle
1952
                   Immediate by the rule SMPAIR.
1953
              Case
1954
                   (3) \mathbb{M}\mathbf{s} = \langle M, \mathbb{M}\mathbf{s} \rangle
1955
1956
                   Immediate by the rule SMPAIR.
              Case
1957
                   (4) \mathbb{M}s = ilet X := \mathbb{M}s in e
1958
1959
                   Immediate by the rule SMLET.
1960
```

```
1962
                    (5) Ms =
                             ilet X \coloneqq \mathbb{M}s in
1963
                             mlet X \coloneqq E in
1964
                             rlet X := R in e
1965
1966
                    Immediate by the rule SRLET.
1967
1968
             LEMMA 5 (COMPOSITIONALITY FOR R).
1969
              For all R, R', and \mathbb{R}s, if \llbracket R \rrbracket = \llbracket R' \rrbracket then \llbracket \mathbb{R}s[R] \rrbracket = \llbracket \mathbb{R}s[R'] \rrbracket.
1970
1971
         Proof.
1972
         Induction on \mathbb{R}s:
1973
               Case
1974
                    (1) \mathbb{R}s = []
1975
                    Immediate.
1976
               Case
                    (2) \mathbb{R}\mathbf{s} = \langle \mathbb{R}\mathbf{s}, R \rangle
1977
                    Immediate by the rule SRPAIR.
1978
1979
               Case
                    (3) \mathbb{R}s = \langle R, \mathbb{R}s \rangle
1980
                    Immediate by the rule SRPAIR.
1981
               Case
1982
                    (4) Rs =
1983
                             ilet X \coloneqq M in
1984
                             mlet X := E in
1985
                             rlet X \coloneqq \mathbb{R}s in e
1986
                    Immediate by the rule SRLET.
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
```

Case

10	4.3 Soundness of Fusion
11	Theorem 21 (Semantics-preserving Fusion for r).
12	For all r_1 and r_2 , if $r_1 \Rightarrow_r r_2$ then $\llbracket r_1 \rrbracket = \llbracket r_2 \rrbracket$.
13	Proof.
14 15	Case analysis on $r_1 \Rightarrow_r r_2$:
16	Case rule FMINR:
17	Immediate from Lemma 6 and Lemma 4.
.8	Case rule FVRED:
0	Immediate the rules SVRED and SMLET on r_1 and SRLET on r_2 .
1 2	Case rule FLETSBIN:
3	By the rules SRLet, SMPAIR, SEEPAIR, SRPAIR, SEBIN.
ł	Similar to Lemma 6, the case for FILETBIN.
	Case rule FMInLets:
7	Immediate from Lemma 7, Lemma 4 and SRLET.
	Case rule FRINLETS:
<i>)</i>	Immediate from Lemma 8, Lemma 5 and SRLET.
, l	
2	Lemma 6 (Semantics-preserving Fusion for m).
3	For all m_1 and m_2 , if $m_1 \Rightarrow_m m_2$ then $\llbracket m_1 \rrbracket = \llbracket m_2 \rrbracket$.
4	Proof.
5	Induction on $m_1 \Rightarrow_m m_2$:
, ,	Case rule FMInM:
	Immediate from the induction hypothesis and Lemma 3.
	Case rule FPNest:
	(1) $s = \mathcal{R} \mathcal{F}(p)$
	$p \in \operatorname{args} \mathcal{R}' \mathcal{F}'(p')$ $p' \in P$
	(2) $s' = \text{ilet } \langle x, x' \rangle \coloneqq \underset{p' \in P}{\mathcal{R}''} \mathcal{F}''(p') \text{ in } x'$
	(3) $\mathcal{R}' \in \{\min, \max\}$
	(4) $f'' \coloneqq \lambda p. \langle \mathcal{F}'(p), \mathcal{F}(p) \rangle$
7	(5) $\mathcal{R}''(\langle a, b \rangle, \langle a', b' \rangle) \coloneqq$
2	if $(a' = a)$ then $\langle a, \mathcal{R}(b, b') \rangle$
2 2	else if $(\mathcal{R}'(a, a') = a)$ then $\langle a, b \rangle$ else $\langle a', b' \rangle$
0	By the rules SPRED and SARGSR on [1],
l	$(6) [[s]] = [\mathbf{v} \mapsto \mathcal{R} \{\mathcal{F}(p) \mid p \in \{p \mid p \in \{\overline{p}\} \land \mathcal{F}'(p) = \mathcal{R}' \{\mathcal{F}'(p) \mid p' \in \{\overline{p}\}\}\}]_{\mathbf{v} \in V(g)}$
2	where
3	(7) $\{\overline{p}\} = P (g)(\mathbf{v})$
ł	(8) $\mathcal{R}' \in \{\min, \max\}$
	By the rules SMLET and SPRED on [2],
	(9) $[\![s']\!] = [v \mapsto \text{second}(\mathcal{R}'' \{\mathcal{F}''(p) \mid p \in [\![P]\!](g)(v)\})]_{v \in V(g)}$
	From [7] and [9],
5	

2059	(10) $[\![s']\!] = \overline{[v \mapsto \mathcal{R}'' \{\mathcal{F}''(p) \mid p \in \{\overline{p}\}\}]}_{v \in \mathcal{V}(q)}$
2060	From [6] and [10], we need to show that for all <i>P</i> ,
2061	(11) $\mathcal{R} \left\{ \mathcal{F}(p) \mid p \in P \land \mathcal{F}'(p) = \mathcal{R}' \left\{ \mathcal{F}'(p) \mid p' \in P \right\} \right\} =$
2062	second $(\mathcal{R}'' \{ \mathcal{F}''(p) \mid p \in P \})$
2063	The proof is by by induction on <i>P</i> .
2064	Base Case:
2065	(12) $P = \{p^*\}$
2066	Form [12],
2067	(13) $\mathcal{R} \left\{ \mathcal{F}(p) \mid p \in P \land \mathcal{F}'(p) = \mathcal{R}' \left\{ \mathcal{F}'(p) \mid p' \in P \right\} \right\} =$
2068	$\mathcal{R} \left\{ \mathcal{F}(p) \mid p \in \{p^*\} \land \mathcal{F}'(p) = \mathcal{R}' \left\{ \mathcal{F}'(p) \mid p' \in \{p^*\} \right\} \right\} =$
2069	$\mathcal{R} \left\{ \mathcal{F}(p) \mid p \in \{p^*\} \land \mathcal{F}'(p) = \mathcal{F}'(p^*) \right\} =$
2070	$\mathcal{F}(p^*)$
2071	Form [12] and [4],
2072	(14) second $(\mathcal{R}'' \ \{\mathcal{F}''(p) \mid p \in P\}) =$
2073	second $(\langle \mathcal{F}'(p^*), \mathcal{F}(p^*) \rangle) =$
2074	$\mathcal{F}(p^*)$
2075	The conclusion is immediate from [13] and [14],
2076	
2077	Inductive Case:
2078	(15) $P = P' \cup \{p^*\}$
2079	Induction Hypothesis:
2080	(16) $\mathcal{R} \left\{ \mathcal{F}(p) \mid p \in P' \land \mathcal{F}'(p) = \mathcal{R}' \left\{ \mathcal{F}'(p) \mid p' \in P' \right\} \right\} =$
2081	second $(\mathcal{R}'' \ \{\mathcal{F}''(p) \mid p \in P'\})$
2082	We show that
2083	$\mathcal{R} \ \{\mathcal{F}(p) \mid p \in P' \cup \{p^*\} \land \mathcal{F}'(p) = \mathcal{R}' \ \{\mathcal{F}'(p) \mid p' \in P' \cup \{p^*\}\}\} =$
2084	second $(\mathcal{R}'' \ \{\mathcal{F}''(p) \mid p \in P' \cup \{p^*\}\})$
2085	That is
2086	$\mathcal{R} \left\{ \mathcal{F}(p) \mid p \in P' \cup \{p^*\} \land \mathcal{F}'(p) = \mathcal{R}' \left(\mathcal{R}' \left\{ \mathcal{F}'(p) \mid p' \in P' \right\}, \mathcal{F}'(p^*) \right) \right\} =$
2087	second $(\mathcal{R}'' (\mathcal{R}'' \ \{\mathcal{F}''(p) \mid p \in P'\}, \mathcal{F}''(p^*)))$
2088	From [5], By induction on S , it can be proved that
2089	(17) $\forall S. \operatorname{first}(\mathcal{R}'' S) = \mathcal{R}' \{a \mid \langle a, a' \rangle \in S\}$
2090	We consider two cases:
2091	Case
2092	(18) $\mathcal{F}'(p^*) = \mathcal{R}' \{ \mathcal{F}'(p) \mid p' \in P' \}$
2095	From [18],
2094	(19) $\mathcal{R} \left\{ \mathcal{F}(p) \mid p \in P' \cup \left\{ p^* \right\} \land \mathcal{F}'(p) = \mathcal{R}' \left(\mathcal{R}' \left\{ \mathcal{F}'(p) \mid p' \in P' \right\}, \mathcal{F}'(p^*) \right) \right\} =$
2095	$\mathcal{R} \left\{ \mathcal{F}(p) \mid p \in P' \cup \{p^*\} \land \mathcal{F}'(p) = \mathcal{F}'(p^*) = \{\mathcal{F}'(p) \mid p' \in P'\} \right\} =$
2090	$\mathcal{R}\left(\mathcal{R}\left\{\mathcal{F}(p) \mid p \in P' \cup \{p^*\} \land \mathcal{F}'(p) = \{\mathcal{F}'(p) \mid p' \in P'\}\right\}, \mathcal{F}(p^*)\right)$
2007	From [18] and [17], $\pi = 1000$
2070	(20) first($\mathcal{R}'' \{\mathcal{F}''(p) \mid p \in P'\}$) = $\mathcal{F}'(p^*)$
2100	We have $(\mathcal{D}''_{\mathcal{D}}, (\mathcal{D}''_{\mathcal{D}}, (\mathcal{D}''_{\mathcal{D}, (\mathcal{D}''_{\mathcal{D}}, (\mathcal{D}''_{\mathcal{D}}, (\mathcal{D}''_{\mathcal{D}, (\mathcal{D}''_{\mathcal{D}, (\mathcal{D}''$
2100	(21) second $(\mathcal{R}'' (\mathcal{R}'' \{\mathcal{F}''(p) \mid p \in P'\}, \mathcal{F}''(p^*))) =$
2102	By [4], $ \langle \mathcal{D} \langle \mathcal{D} \langle \mathcal{D} \langle \mathcal{D} \langle \mathcal{D} \rangle = \mathbb{P}(1) \langle \mathcal{D} \langle \mathcal{D} \langle \mathcal{D} \langle \mathcal{D} \langle \mathcal{D} \rangle = \mathbb{P}(1) \langle \mathcal{D} \langle$
2103	second $(\mathcal{K}''(\mathcal{K}'' \{\mathcal{F}''(p) \mid p \in P'\}, \langle \mathcal{F}'(p^*), \mathcal{F}(p^*) \rangle)) =$
2104	By [5] and [20], $d(\pi')(\pi^*) \oplus (\pi - \pi - \pi)(\pi')(\pi')(\pi')(\pi')(\pi')(\pi')(\pi')(\pi')(\pi')(\pi$
2105	second $(\langle \mathcal{F}^{\prime}(p) \rangle, \mathcal{K} ($ second $(\mathcal{K}^{\prime\prime} \{\mathcal{F}^{\prime\prime}(p) \mid p \in F \}), \mathcal{F} (p^{\prime}) \rangle)) = \mathcal{O}($
2106	κ (second ($\kappa = \{\mathcal{F} \mid p\} \mid p \in P\}$), $\mathcal{F}(p^{-})$)
2107	

2108	Thus
2109	(22) second $(\mathcal{R}''(\mathcal{R}'' \{\mathcal{F}''(p) \mid p \in P'\}, \mathcal{F}''(p^*))) =$
2110	\mathcal{R} (second (\mathcal{R}'' { $\mathcal{F}''(p) \mid p \in P'$ }), $\mathcal{F}(p^*)$)
2111	From [19] and [22], we have the conclusion:
2112	$\mathcal{R} \ \left\{ \mathcal{F}(p) \mid p \in P' \cup \{p^*\} \land \mathcal{F}'(p) = \mathcal{R}' \left(\mathcal{R}' \ \left\{ \mathcal{F}'(p) \mid p' \in P' \right\}, \mathcal{F}'(p^*) \right) \right\} =$
2113	second $(\mathcal{R}''(\mathcal{R}'' \in \mathcal{F}''(p) \mid p \in P'), \mathcal{F}''(p^*)))$
2114	Case
2115	(23) $\mathcal{F}'(p^*) \neq \mathcal{R}' \{ \mathcal{F}'(p) \mid p' \in P' \}$
2116	We assume $\mathcal{R}' = \max$. The other case $\mathcal{R}' = \min$ is similar.
2117	We consider two sub-cases.
2118	Sub-case
2119	(24) $\mathcal{R}'(\mathcal{R}' \{\mathcal{F}'(p) \mid p' \in P'\}, \mathcal{F}'(p^*)) = \mathcal{F}'(p^*)$
2120	From [23] and [24],
2121	(25) $\forall p' \in P'. \mathcal{F}'(p') < \mathcal{F}'(p^*)$
2122	We have
2123	$(26) \mathcal{R} \left\{ \mathcal{F}(p) \mid p \in P' \cup \{p^*\} \land \mathcal{F}'(p) = \mathcal{R}' \left(\mathcal{R}' \left\{ \mathcal{F}'(p) \mid p' \in P' \right\}, \mathcal{F}'(p^*) \right) \right\} = \mathcal{R}' \left(\mathcal{R}' \left\{ \mathcal{F}'(p) \mid p' \in P' \right\}, \mathcal{F}'(p^*) \right) = \mathcal{R}' \left(\mathcal{R}' \left\{ \mathcal{F}'(p) \mid p' \in P' \right\}, \mathcal{F}'(p^*) \right) = \mathcal{R}' \left(\mathcal{R}' \left\{ \mathcal{F}'(p) \mid p' \in P' \right\}, \mathcal{F}'(p^*) \right) = \mathcal{R}' \left(\mathcal{R}' \left\{ \mathcal{F}'(p) \mid p' \in P' \right\}, \mathcal{F}'(p^*) \right) = \mathcal{R}' \left(\mathcal{R}' \left\{ \mathcal{F}'(p) \mid p' \in P' \right\}, \mathcal{F}'(p^*) \right) = \mathcal{R}' \left(\mathcal{R}' \left\{ \mathcal{F}'(p) \mid p' \in P' \right\}, \mathcal{F}'(p^*) \right) = \mathcal{R}' \left(\mathcal{R}' \left\{ \mathcal{F}'(p) \mid p' \in P' \right\}, \mathcal{F}'(p^*) \right) = \mathcal{R}' \left(\mathcal{R}' \left\{ \mathcal{F}'(p) \mid p' \in P' \right\}, \mathcal{F}'(p^*) \right) = \mathcal{R}' \left(\mathcal{R}' \left\{ \mathcal{F}'(p) \mid p' \in P' \right\}, \mathcal{F}'(p^*) \right) = \mathcal{R}' \left(\mathcal{R}' \left\{ \mathcal{F}'(p) \mid p' \in P' \right\}, \mathcal{F}'(p^*) \right) = \mathcal{R}' \left(\mathcal{R}' \left\{ \mathcal{F}'(p) \mid p' \in P' \right\}, \mathcal{F}'(p^*) \right) = \mathcal{R}' \left(\mathcal{R}' \left\{ \mathcal{F}'(p) \mid p' \in P' \right\}, \mathcal{F}'(p^*) \right) = \mathcal{R}' \left(\mathcal{R}' \left\{ \mathcal{F}'(p) \mid p' \in P' \right\}, \mathcal{F}'(p^*) \right) = \mathcal{R}' \left(\mathcal{R}' \left\{ \mathcal{F}'(p) \mid p' \in P' \right\}, \mathcal{F}'(p^*) \right) = \mathcal{R}' \left(\mathcal{R}' \left\{ \mathcal{F}'(p) \mid p' \in P' \right\}, \mathcal{F}'(p^*) \right) = \mathcal{R}' \left(\mathcal{R}' \left\{ \mathcal{F}'(p) \mid p' \in P' \right\}, \mathcal{F}'(p^*) \right) = \mathcal{R}' \left(\mathcal{R}' \left\{ \mathcal{F}'(p) \mid p' \in P' \right\}, \mathcal{F}'(p^*) \right) = \mathcal{R}' \left(\mathcal{R}' \left\{ \mathcal{F}'(p) \mid p' \in P' \right\}, \mathcal{F}'(p^*) \right) = \mathcal{R}' \left(\mathcal{R}' \left\{ \mathcal{F}'(p) \mid p' \in P' \right\}, \mathcal{F}'(p^*) \right) = \mathcal{R}' \left(\mathcal{R}' \left\{ \mathcal{F}'(p) \mid p' \in P' \right\}, \mathcal{F}'(p^*) \right) = \mathcal{R}' \left(\mathcal{R}' \left\{ \mathcal{F}'(p) \mid p' \in P' \right\}, \mathcal{F}'(p^*) \right) = \mathcal{R}' \left(\mathcal{R}' \left\{ \mathcal{R}' \left\{ \mathcal{F}'(p) \mid p' \in P' \right\}, \mathcal{F}'(p^*) \right\}$
2124	By [24]
2125	$\mathcal{R} \ \{\mathcal{F}(p) \mid p \in P' \cup \{p^*\} \land \mathcal{F}'(p) = \mathcal{F}'(p^*)\} =$
2126	By [25]
2127	$\mathcal{F}(p^*)$
2128	Thus
2129	$(27) \mathcal{R} \left\{ \mathcal{F}(p) \mid p \in P' \cup \{p^*\} \land \mathcal{F}'(p) = \mathcal{R}' \left(\mathcal{R}' \left\{ \mathcal{F}'(p) \mid p' \in P' \right\}, \mathcal{F}'(p^*) \right) \right\} = \mathcal{R}' \left(\mathcal{R}' \left\{ \mathcal{F}'(p) \mid p' \in P' \right\}, \mathcal{F}'(p^*) \right) = \mathcal{R}' \left(\mathcal{R}' \left\{ \mathcal{F}'(p) \mid p' \in P' \right\}, \mathcal{F}'(p^*) \right) = \mathcal{R}' \left(\mathcal{R}' \left\{ \mathcal{F}'(p) \mid p' \in P' \right\}, \mathcal{F}'(p^*) \right) = \mathcal{R}' \left(\mathcal{R}' \left\{ \mathcal{F}'(p) \mid p' \in P' \right\}, \mathcal{F}'(p^*) \right) = \mathcal{R}' \left(\mathcal{R}' \left\{ \mathcal{F}'(p) \mid p' \in P' \right\}, \mathcal{F}'(p^*) \right) = \mathcal{R}' \left(\mathcal{R}' \left\{ \mathcal{F}'(p) \mid p' \in P' \right\}, \mathcal{F}'(p^*) \right) = \mathcal{R}' \left(\mathcal{R}' \left\{ \mathcal{F}'(p) \mid p' \in P' \right\}, \mathcal{F}'(p^*) \right) = \mathcal{R}' \left(\mathcal{R}' \left\{ \mathcal{F}'(p) \mid p' \in P' \right\}, \mathcal{F}'(p^*) \right) = \mathcal{R}' \left(\mathcal{R}' \left\{ \mathcal{F}'(p) \mid p' \in P' \right\}, \mathcal{F}'(p^*) \right) = \mathcal{R}' \left(\mathcal{R}' \left\{ \mathcal{F}'(p) \mid p' \in P' \right\}, \mathcal{F}'(p^*) \right) = \mathcal{R}' \left(\mathcal{R}' \left\{ \mathcal{F}'(p) \mid p' \in P' \right\}, \mathcal{F}'(p^*) \right) = \mathcal{R}' \left(\mathcal{R}' \left\{ \mathcal{F}'(p) \mid p' \in P' \right\}, \mathcal{F}'(p^*) \right) = \mathcal{R}' \left(\mathcal{R}' \left\{ \mathcal{F}'(p) \mid p' \in P' \right\}, \mathcal{F}'(p^*) \right) = \mathcal{R}' \left(\mathcal{R}' \left\{ \mathcal{F}'(p) \mid p' \in P' \right\}, \mathcal{F}'(p^*) \right) = \mathcal{R}' \left(\mathcal{R}' \left\{ \mathcal{F}'(p) \mid p' \in P' \right\}, \mathcal{F}'(p^*) \right) = \mathcal{R}' \left(\mathcal{R}' \left\{ \mathcal{F}'(p) \mid p' \in P' \right\}, \mathcal{F}'(p^*) \right) = \mathcal{R}' \left(\mathcal{R}' \left\{ \mathcal{F}'(p) \mid p' \in P' \right\}, \mathcal{F}'(p^*) \right) = \mathcal{R}' \left(\mathcal{R}' \left\{ \mathcal{F}'(p) \mid p' \in P' \right\}, \mathcal{F}'(p^*) \right) = \mathcal{R}' \left(\mathcal{R}' \left\{ \mathcal{F}'(p) \mid p' \in P' \right\}, \mathcal{F}'(p^*) \right) = \mathcal{R}' \left(\mathcal{R}' \left\{ \mathcal{F}'(p) \mid p' \in P' \right\}, \mathcal{F}'(p^*) \right) = \mathcal{R}' \left(\mathcal{R}' \left\{ \mathcal{F}'(p) \mid p' \in P' \right\}, \mathcal{F}'(p^*) \right) = \mathcal{R}' \left(\mathcal{R}' \left\{ \mathcal{R}' \left$
2130	$\mathcal{F}(p^*)$
2131	From [18] and [23], $\pi(2) = \pi(2) = \pi(2)$
2132	(28) first($\mathcal{R}'' \ \{\mathcal{F}''(p) \mid p \in P'\} \} \neq \mathcal{F}'(p^*)$
2133	We have $I(\mathcal{O} \mathcal{U} (\mathcal{O} ($
2134	(29) second $(\mathcal{K}^{(r)}(\mathcal{K}^{(r)} \{\mathcal{F}^{(r)}(p) \mid p \in P^{r}\}, \mathcal{F}^{(r)}(p^{r}))) =$
2135	By [4], $(\mathcal{O}''_{\mathcal{O}}(\mathcal{O}''_{\mathcal{O}}(\tau) \mid \tau \in \mathcal{D}') \mid (\mathcal{O}''_{\mathcal{O}}(\tau^*) \mid \mathcal{O}(\tau^*)))$
2136	second $(\mathcal{K} \ (\mathcal{K} \ \{\mathcal{F} \ (p) \mid p \in P\}, (\mathcal{F} \ (p), \mathcal{F} \ (p)))) =$
2137	By [5], [28] and [24], $\mathcal{F}'(r^*) \mathcal{F}(r^*)$
2130	second $(\langle \mathcal{F} (p), \mathcal{F} (p) \rangle) = \mathcal{F}(p^*)$
2139	f(p)
2140	1 mus (30) cocond $(\mathcal{P}''(\mathcal{P}''(\mathcal{F}''(p) \mid p \in \mathcal{P}') \mid \mathcal{F}''(p^*))) =$
2141	$(50) \text{ second } (\mathcal{K} (\mathcal{K} \{\mathcal{I} (p) \mid p \in \mathcal{I}\}, \mathcal{I} (p))) = \mathcal{F}(p^*)$
2142	f(p) From [27] and [30], we have the conclusion:
2143	$\mathcal{R} \{\mathcal{F}(p) \mid p \in \mathcal{P}' \mid \{p^*\} \land \mathcal{F}'(p) - \mathcal{R}'(\mathcal{R}') \mid p' \in \mathcal{P}'\} \mathcal{F}'(p^*)\} = \mathcal{R}'(\mathcal{R}')$
2145	$\mathcal{R} \left(p \left(p \right) \mid p \in I \bigcirc \left(p \mid p \in I) \right) \right) \right) \right) \right) \right) \right)$
2146	Sub-case
2147	(31) $\mathcal{R}'(\mathcal{R}' \{\mathcal{F}'(p) \mid p' \in P'\} \mid \mathcal{F}'(p^*)) = \mathcal{R}'\{\mathcal{F}'(p) \mid p' \in P'\}$
2148	This sub-case is similar to the previous sub-case
2149	$\mathcal{F}'(p^*)$ and $\mathcal{R}' \{\mathcal{F}'(p) \mid p' \in P'\}$ replace each other
2150	(p) (p) and (r) (p) (
2151	Case rule FPRED:
2152	Immediate from the rules SMLET and SMM.
2153	
2154	Case rule FILETBIN:
2155	(32) $s = (\text{ilet } X_1 := M_1 \text{ in } e_1) \oplus (\text{ilet } X_2 := M_2 \text{ in } e_2)$
2156	(, (,)) = (,)

(33) $s' = \text{ilet} \langle X_1, X_2 \rangle := \langle M_1, M_2 \rangle \text{ in } e_1 \oplus e_2$ 2157 2158 (34) free $(e_1) \cap X_2 = \emptyset$ 2159 (35) free $(e_2) \cap X_1 = \emptyset$ 2160 By rules SMBIN and SMLET on [32], we have 2161 $(36) [[s]] = [\mathbf{v} \mapsto [[e_1 [X_1 \coloneqq [[M_1]] (g)(\mathbf{v})]]] \oplus [[e_2 [X_2 \coloneqq [[M_2]] (g)(\mathbf{v})]]]]_{\mathbf{v} \in \mathsf{V}(a)}$ 2162 By rules SMLET on [33], we have 2163 $(37) [[s']] = \overline{[\mathbf{v} \mapsto} [(e_1 \oplus e_2) [\langle X_1, X_2 \rangle \coloneqq [\langle M_1, M_2 \rangle] (g)(\mathbf{v})]]]_{\mathbf{v} \in V(q)}$ 2164 By [37] and the rule SMPAIR, we have 2165 $(38) \quad [\![s']\!] = \overline{[\mathsf{v} \mapsto [\![(e_1 \oplus e_2)]\!](X_1, X_2)} \coloneqq \langle [\![M_1]\!](g)(\mathsf{v}), [\![M_2]\!](g)(\mathsf{v})\rangle]]\!]_{\mathsf{v} \in \mathsf{V}(q)}$ 2166 From [38], and the rule SEBIN, we have 2167 $(39) [[s']] = [v \mapsto [[e_1[\langle X_1, X_2 \rangle := \langle [[M_1]](q)(v), [[M_2]](q)(v) \rangle]]] \oplus$ 2168 $\overline{\llbracket e_2 \llbracket \langle X_1, X_2 \rangle \coloneqq \langle \llbracket M_1 \rrbracket (g)(\mathsf{v}), \llbracket M_2 \rrbracket (g)(\mathsf{v}) \rangle \rrbracket}_{\mathsf{v} \in \mathsf{V}(q)}$ 2169 2170 From [39], [34] and [35], we have $(40) \quad \llbracket s' \rrbracket = \overline{\left[\mathsf{v} \mapsto \llbracket e_1 \left[X_1 \coloneqq \llbracket M_1 \rrbracket \left(g \right)(\mathsf{v}) \right] \rrbracket \oplus \llbracket e_2 \left[X_2 \coloneqq \llbracket M_2 \rrbracket \left(g \right)(\mathsf{v}) \right] \rrbracket \right]}_{\mathsf{v} \in \mathsf{V}(a)}$ 2171 2172 From [36] and [40], we have 2173 $\llbracket s \rrbracket = \llbracket s' \rrbracket$ 2174 2175 Case rule FMINILET: 2176 Immediate from Lemma 7, Lemma 4 and SMLET. 2177 2178 LEMMA 7 (SEMANTICS-PRESERVING FUSION FOR M). 2179 For all M_1 and M_2 , if $M_1 \Rightarrow_M M_2$ then $\llbracket M_1 \rrbracket = \llbracket M_2 \rrbracket$. 2180 2181 Proof. 2182 Induction on $M_1 \Rightarrow_M M_2$: 2183 Case rule FMPAIR: 2184 (1) $M_1 = \langle \mathcal{R} \mathcal{F}, \mathcal{R}' \mathcal{F}' \rangle$ 2185 (2) $M_2 = \mathcal{R}'' F''$ 2186 (3) $f'' \coloneqq \lambda p. \langle \mathcal{F}'(p), \mathcal{F}(p) \rangle$ 2187 (4) $\mathcal{R}''(\langle a, b \rangle, \langle a', b' \rangle) \coloneqq \langle \mathcal{R}(a, a'), \mathcal{R}'(b, b') \rangle$ 2188 By SMPAIR, SMM and SPRED on [1], we have 2189 (5) $\llbracket M_1 \rrbracket = \left\langle \overline{[\mathbf{v} \mapsto \mathcal{R} \{\mathcal{F}(p) \mid p \in \llbracket \text{Paths} \rrbracket (\mathbf{v})\}]}_{\mathbf{v} \in V(q)}, \overline{[\mathbf{v} \mapsto \mathcal{R}' \{\mathcal{F}'(p) \mid p \in \llbracket \text{Paths} \rrbracket (\mathbf{v})\}]}_{\mathbf{v} \in V(q)} \right\rangle$ 2190 By SMM and SPRED on [2] and [3] and [4], we have 2191 (6) $\llbracket M_2 \rrbracket = [\mathbf{v} \mapsto \mathcal{R}'' \{ \langle \mathcal{F}(p), \mathcal{F}'(p) \rangle \mid p \in \llbracket \text{Paths} \rrbracket (\mathbf{v}) \}]_{\mathbf{v} \in V(q)}$ 2192 2193 By [6] and [4], we have (7) $[\![M_2]\!] = \left\langle \overline{[\mathbf{v} \mapsto \mathcal{R} \ \{\mathcal{F}(p) \mid p \in [\![\operatorname{Paths}]\!](\mathbf{v})\}]}_{\mathbf{v} \in \mathsf{V}(g)}, \overline{[\mathbf{v} \mapsto \mathcal{R}' \ \{\mathcal{F}'(p) \mid p \in [\![\operatorname{Paths}]\!](\mathbf{v})\}]}_{\mathbf{v} \in \mathsf{V}(q)} \right\rangle$ 2194 2195 From [5] and [7], we have 2196 (8) $[\![M_1]\!] = [\![M_2]\!]$ 2197 2198 2199 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 $\left[R_1[X \coloneqq \overline{d}] \right] = \left[R_2[X \coloneqq \overline{d}] \right]$. 2200 2201 Proof. 2202 2203 Induction on $R_1 \implies_R R_2$: Case rule FRPAIR: 2204 2205

2206	(1) $R_1 = \langle \mathcal{R}_1 x_1, \mathcal{R}_2 x_2 \rangle$
2207	(2) $R_2 = \mathcal{R}_3 \langle x_1, x_2 \rangle$
2208	(3) $\mathcal{R}_3(\langle a, b \rangle, \langle a', b' \rangle) \coloneqq \langle \mathcal{R}_1(a, a'), \mathcal{R}_2(b, b') \rangle$
2209	If x_1 or $x_2 \notin X$, $x_1[X := \overline{d}] = \bot$ or $x_2[X := \overline{d}] = \bot$
2210	as the rule SRR is the only semantic rule for <i>R</i> .
2211	(4) $\begin{bmatrix} P_{1}[X := \overline{d}] \end{bmatrix} = \begin{bmatrix} P_{2}[X := \overline{d}] \end{bmatrix} = 1$
2212	(f) $\begin{bmatrix} R_1 \begin{bmatrix} X & -u \end{bmatrix} \end{bmatrix} = \begin{bmatrix} R_2 \begin{bmatrix} X & -u \end{bmatrix} \end{bmatrix} = \pm$.
2213	I hus, the remained case is that (5) is the function of V is \overline{V}
2214	(5) $x_1 := [v \mapsto n_v]_{v \in V(g)} \in X := a$
2215	(6) $x_2 \coloneqq [\mathbf{v} \mapsto n'_{\mathbf{v}}]_{\mathbf{v} \in \mathbf{V}(g)} \in X \coloneqq d$
2216	From [1], [5] and [6], we have
2217	(7) $R_1 = \left\langle \mathcal{R}_1 \left[\mathbf{v} \mapsto n_{\mathbf{v}} \right]_{\mathbf{v} \in \mathbf{V}(g)}, \mathcal{R}_2 \left[\mathbf{v} \mapsto n'_{\mathbf{v}} \right]_{\mathbf{v} \in \mathbf{V}(g)} \right\rangle$
2218	From [2], [5] and [6], we have
2219	(8) $R_2 = \mathcal{R}_3 \left\langle \overline{[\mathbf{v} \mapsto n_{\mathbf{v}}]}_{\mathbf{v} \in \mathcal{V}(q)}, \overline{[\mathbf{v} \mapsto n'_{\mathbf{v}}]}_{\mathbf{v} \in \mathcal{V}(q)} \right\rangle$
2220	By SRPAIR SRR and SVRED on [7] we have
2222	$(0) [P] = \langle \mathcal{P} (\overline{\mathbf{r}}) \rangle \langle \mathcal{P} (\overline{\mathbf{r}}') \rangle \langle \mathcal{P} (\overline{\mathbf{r}'}) \rangle \langle \mathcal{P} (\overline{\mathbf{r}}') \rangle \langle \mathcal{P} (\overline{\mathbf{r}'}) \rangle \langle \mathcal{P} (\mathbf{r$
2223	$(9) [K_1] = \langle X_1 \{ n_{V \ V \in V(g)} \}, X_2 \{ n_{V \ V \in V(g)} \} \rangle$
2224	By SRPAIR, SRR and SVRED on [8], we have
2225	(10) $\llbracket R_2 \rrbracket = \mathcal{R}_3 \left\{ \langle n_v, n'_v \rangle_{v \in V(g)} \right\}$
2226	From [10] and [3], we have
2227	(11) $\llbracket R_2 \rrbracket = \langle \mathcal{R}_1 \{ \overline{n_v} \}_{v \in V(g)}, \mathcal{R}_2 \{ \overline{n'_v} \}_{v \in V(g)} \rangle$
2228	From [9] and [11], we have
2229	$\left\ R_1[X \coloneqq \overline{d}] \right\ = \left\ R_2[X \coloneqq \overline{d}] \right\ .$
2230	
2231	
2232	
2233	
2234	
2235	
2236	
2237	
2238	
2239	
2240	
2241	
2243	
2244	
2245	
2246	
2247	

2255 4.4 Iteration Correctness Conditions

2256 4.4.1 Pull, Idempotent 2257 THEOREM 22 (CORRECTNESS OF PULL (IDEMPOTENT REDUCTION)). 2258 For all $\mathcal{R}, \mathcal{F}, C, I, \mathcal{P}$, and $k \geq 1$, if the conditions $\mathbb{C}_1 - \mathbb{C}_9$ hold, 2259 $\mathcal{S}_{\text{pull+}}^k(v) = \mathcal{S}pec^k(v)$ 2260 2261 We assume that 2262 (1) $\forall n. \mathcal{R}(n, \perp) = n$ 2263 (2) $\forall n, n'. \mathcal{R}(n, n') = \mathcal{R}(n', n)$ 2264 (3) $\forall n, n', n''$. $\mathcal{R}(\mathcal{R}(n, n'), n'') = \mathcal{R}(n, \mathcal{R}(n', n''))$ 2265 (4) $\forall n. \mathcal{R}(n, n) = n$ 2266 (5) $\forall v \in V. C(\langle v, v \rangle) \rightarrow I(v) = \mathcal{F}(\langle v, v \rangle)$ 2267 (6) $\forall v \in V. \neg C(\langle v, v \rangle) \rightarrow I(v) = \bot$ 2268 (7) $\forall e. \mathcal{P}(\bot, e) = \bot$ 2269 2270 (8) $\forall p_1, p_2 \in \mathsf{P}, v \in \mathsf{V}.$ 2271 $tail(p_1) = tail(p_2) \rightarrow$ 2272 let $u := tail(p_1)$ in 2273 $\mathcal{P}\left[\mathcal{R}(\mathcal{F}(p_1), \mathcal{F}(p_2)), \langle u, v \rangle\right] =$ 2274 $\mathcal{R}\left[\mathcal{F}(p_1 \cdot \langle u, v \rangle), \, \mathcal{F}(p_2 \cdot \langle u, v \rangle)\right]$ 2275 (9) $\forall p, e. \mathcal{P}(\mathcal{F}(p), e) = \mathcal{F}(p \cdot e)$ 2276 Form Def. 6, we have 2277 $Spec^{k}(v) = \mathcal{R}_{p \in \{p \mid p \in Paths(v) \land C(p) \land length(p) < k\}} \mathcal{F}(p)$ 2278 2279 Proof by induction on *k*: 2280 Base Case: 2281 k = 12282 We should show that 2283 $S^{1}_{\mathsf{pull}+}(v) = \mathcal{R}_{p \in \{p \mid p \in \mathsf{Paths}(v) \land C(p) \land \mathsf{length}(p) < 1\}}\mathcal{F}(p)$ 2284 that is 2285 $\mathcal{S}^1_{\mathsf{pull}+}(v) = \mathcal{R}_{p \in \{p \mid p = \langle v, v \rangle \land C(p)\}} \mathcal{F}(p)$ 2286 We consider two cases: 2287 Case: 2288 (10) $C(\langle v, v \rangle)$ 2289 We should show that 2290 $\mathcal{S}^{1}_{\mathsf{pull}+}(v) = \mathcal{R}_{\{\langle v, v \rangle\}} \mathcal{F}(p)$ 2291 that is 2292 $\mathcal{S}^{1}_{\mathsf{pull}+}(v) = \mathcal{F}(\langle v, v \rangle)$ 2293 that is straightforward from Def. 1, [5] and [10]. 2294 Case: 2295 (11) $\neg C(\langle v, v \rangle)$ 2296 We should show that 2297 $\mathcal{S}^1_{\mathsf{pull}+}(v) = \mathop{\mathcal{R}}_{\emptyset} \mathcal{F}(p)$ 2298 that is 2299 $\mathcal{S}^{1}_{\mathsf{pull}+}(v) = \bot$ 2300 that is straightforward from Def. 1, [6] and [11]. 2301 2302 2303

Anon.

Inductive Case: 2304 2305 (12) k > 12306 The induction hypothesis is: (13) $S_{\text{pull}+}^{k'}(v) = \mathcal{R}_{p \in \{p \mid p \in \text{Paths}(v) \land C(p) \land \text{length}(p) < k'\}}\mathcal{F}(p)$ for all v and $k' \le k$ 2307 2308 We should show that $\mathcal{S}_{\mathsf{pull}+}^{k+1}(v) = \mathcal{R}_{p \in \{p \mid p \in \mathsf{Paths}(v) \land C(p) \land \mathsf{length}(p) < k+1\}} \mathcal{F}(p)$ 2309 2310 From Def. 1, we consider two cases: 2311 Case: 2312 (14) $\operatorname{CPreds}^k(v) \neq \emptyset$ 2313 (15) $S_{\text{pull}+}^{k+1}(v) = \mathcal{R}\left[S_{\text{pull}+}^{k}(v), \mathcal{R}_{u \in \text{preds}(v)} \mathcal{P}\left(S_{\text{pull}+}^{k}(u), \langle u, v \rangle\right)\right]$ 2314 From [15] and [13] 2315 (16) $\mathcal{S}_{\text{pull}+}^{k+1}(v) = \mathcal{R}[$ 2316 $\mathcal{S}_{\text{pull}+}^{k}(v),$ 2317 2318 $\mathcal{R}_{u \in \text{preds}(v)} \left[\mathcal{P} \left(\mathcal{R}_{p \in \{p \mid p \in \text{Paths}(u) \land C(p) \land \text{length}(p) < k \}} \mathcal{F}(p), \langle u, v \rangle \right) \right] \right]$ 2319 In the case that the size of the set of paths is more than one, from [8], and 2320 in the case that the set of paths is singleton, from $\mathcal{R}_{\{v\}} = v$ and [9], and 2321 in the case that the set of paths is empty, from $\mathcal{R}_{\emptyset} = \bot$ and [7], 2322 we have 2323 (17) $\mathcal{P}\left(\mathcal{R}_{p \in \{p \mid p \in \text{Paths}(u) \land C(p) \land \text{length}(p) < k\}}\mathcal{F}(p), \langle u, v \rangle\right) =$ 2324 $\mathcal{R}_{p \in \{p \mid p \in \text{Paths}(u) \land C(p) \land \text{length}(p) < k\}} \mathcal{F}(p \cdot \langle u, v \rangle)$ 2325 After substituting [17] in [16] 2326 (18) $\mathcal{S}_{\text{pull}+}^{k+1}(v) = \mathcal{R}[$ 2327 $\mathcal{S}_{\text{pull+}}^{k}(v),$ 2328 $\mathcal{R}_{u \in \text{preds}(v)} \left[\mathcal{R}_{p \in \{p \mid p \in \text{Paths}(u) \land C(p) \land \text{length}(p) < k\}} \mathcal{F}(p \cdot \langle u, v \rangle) \right] \right]$ 2329 that is 2330 (19) $\mathcal{S}_{\text{pull}+}^{k+1}(v) = \mathcal{R}[$ 2331 $\mathcal{S}^k_{\mathsf{pull}+}(v),$ 2332 2333 $\mathcal{R}_{p \in \{p \mid \exists u. \ p \in \mathsf{Paths}(u) \land u \in \mathsf{preds}(v) \land C(p) \land \mathsf{length}(p) < k\}} \mathcal{F}(p \cdot \langle u, v \rangle)]$ 2334 From [19] and Lemma 9 (20) $\mathcal{S}_{\text{pull}+}^{k+1}(v) = \mathcal{R}[$ 2335 2336 $\mathcal{S}_{\text{pull+}}^k(v),$ 2337 $\mathcal{R}_{p \in \{p \mid \exists u, p \in \mathsf{Paths}(u) \land u \in \mathsf{preds}(v) \land C(p \cdot \langle u, v \rangle) \land \mathsf{length}(p) < k\}} \mathcal{F}(p \cdot \langle u, v \rangle)]$ 2338 that is 2339 (21) $S_{\text{pull}+}^{k+1}(v) = \mathcal{R}[$ 2340 $\mathcal{S}_{\text{null}+}^{k}(v),$ 2341 $\mathcal{R}_{p' \in \{p' \mid p' \in \mathsf{Paths}(v) \land C(p') \land 0 < \mathsf{length}(p') < k+1\}} \mathcal{F}(p')]$ 2342 From [21] and [13] 2343 (22) $\mathcal{S}_{\text{push+}}^{k+1}(v) = \mathcal{R}[$ 2344 $\mathcal{R}_{p \in \{p \mid p \in \mathsf{Paths}(v) \land C(p) \land \mathsf{length}(p) < k\}} \mathcal{F}(p),$ 2345 $\mathcal{R}_{p \in \{p \mid p \in \mathsf{Paths}(v) \land C(p) \land 0 < \mathsf{length}(p) < k+1\}} \mathcal{F}(p)]$ 2346 2347 From [22] and [4] $\mathcal{S}_{\text{push+}}^{k+1}(v) = \mathcal{R}_{p \in \{p \mid p \in \text{Paths}(v) \land C(p) \land \text{length}(p) < k+1\}} \mathcal{F}(p),$ 2348 2349 2350 Case: 2351

46

(23) $\operatorname{CPreds}^k(v) = \emptyset$ 2353 (24) $\mathcal{S}_{\text{pull}+}^{k+1}(v) = \mathcal{S}_{\text{pull}+}^{k}(v)$ 2354 2355 (25) $\operatorname{CPreds}^{k}(v) = \left\{ u \mid u \in \operatorname{preds}(v) \land \mathcal{S}_{\operatorname{pull}}^{k}(u) \neq \mathcal{S}_{\operatorname{pull}}^{k-1}(u) \right\}$ 2356 From [13] and [24], 2357 (26) $\mathcal{S}_{\text{pull+}}^{k+1}(v) = \mathcal{R}_{p \in \{p \mid p \in \text{Paths}(v) \land C(p) \land \text{length}(p) < k\}} \mathcal{F}(p)$ 2358 From [26] and [1], 2359 (27) $\mathcal{S}_{\text{pull}+}^{k+1}(v) = \mathcal{R}[$ 2360 2361 $\mathcal{R}_{p \in \{p \mid \exists u. \ p \in \mathsf{Paths}(u) \land u \in \mathsf{preds}(v) \land C(p \cdot \langle u, v \rangle) \land 0 < \mathsf{length}(p \cdot \langle u, v \rangle) < k\}} \mathcal{F}(p),$ 2362 $\mathcal{R}_{p \in \{p \mid p = \langle v, v \rangle \land C(\langle v, v \rangle)\}} \mathcal{F}(p)]$ 2363 From [27] and Lemma 9, (28) $\mathcal{S}_{\text{pull}+}^{k+1}(v) = \mathcal{R}[$ 2364 2365 $\mathcal{R}_{p \in \{p \mid \exists u. \ p \in \mathsf{Paths}(u) \land u \in \mathsf{preds}(v) \land C(p) \land 0 < \mathsf{length}(p \cdot \langle u, v \rangle) < k\}} \mathcal{F}(p),$ 2366 $\mathcal{R}_{p \in \{p \mid p = \langle v, v \rangle \land C(\langle v, v \rangle)\}} \mathcal{F}(p)]$ 2367 From [28], [2] and [3] (29) $\mathcal{S}_{\text{pull}+}^{k+1}(v) = \mathcal{R}[$ 2368 2369 $\mathcal{R}_{u \in \mathsf{preds}(v)} \ \mathcal{R}_{p' \in \{p' \mid \exists u. \ p' \in \mathsf{Paths}(u) \land C(p') \land \mathsf{length}(p') < k-1\}} \mathcal{F}(p' \cdot \langle u, v \rangle),$ 2370 $\mathcal{R}_{p \in \{p \mid p = \langle v, v \rangle \land C(\langle v, v \rangle)\}} \mathcal{F}(p)]$ 2371 In the case that the size of the set of paths is more than one, from [8], and 2372 in the case that the set of paths is singleton, from $\mathcal{R}_{\{v\}} = v$ and [9], and 2373 in the case that the set of paths is empty, from $\mathcal{R}_{\emptyset} = \bot$ and [7], 2374 (30) $\mathcal{P}\left(\mathcal{R}_{p \in \{p \mid p \in \text{Paths}(u) \land C(p) \land \text{length}(p) < k-1\}}\mathcal{F}(p), \langle u, v \rangle\right) =$ 2375 $\mathcal{R}_{p \in \{p \mid p \in \mathsf{Paths}(u) \land C(p) \land \mathsf{length}(p) < k-1\}} \mathcal{F}(p \cdot \langle u, v \rangle)$ 2376 From [29] and [30], we have 2377 (31) $\mathcal{S}_{\text{pull}+}^{k+1}(v) = \mathcal{R}[$ 2378 $\mathcal{R}_{u \in \text{preds}(v)} \mathcal{P}\left(\mathcal{R}_{p \in \{p \mid p \in \text{Paths}(u) \land C(p) \land \text{length}(p) < k-1\}} \mathcal{F}(p), \langle u, v \rangle\right),$ 2379 $\mathcal{R}_{p \in \{p \mid p = \langle v, v \rangle \land C(\langle v, v \rangle)\}} \mathcal{F}(p)]$ 2380 From [31] and [13], we have (32) $\mathcal{S}_{\text{pull}+}^{k+1}(v) = \mathcal{R}[$ 2381 2382 $\mathcal{R}_{u \in \text{preds}(v)} \mathcal{P}\left(S_{\text{pull}+}^{k-1}(u), \langle u, v \rangle\right),$ 2383 $\mathcal{R}_{p \in \{p \mid p = \langle v, v \rangle \land C(\langle v, v \rangle)\}} \mathcal{F}(p)]$ 2384 2385 From [23] and [25], we have 2386 (33) For all $u \in \operatorname{preds}(v)$: $S_{\operatorname{pull}+}^{k}(u) = S_{\operatorname{pull}+}^{k-1}(u)$ 2387 From [32] and [33], we have 2388 (34) $\mathcal{S}_{\text{pull}+}^{k+1}(v) = \mathcal{R}[$ 2389 $\mathcal{R}_{u \in \text{preds}(v)} \mathcal{P}\left(S_{\text{null}+}^{k}(u), \langle u, v \rangle\right),$ 2390 2391 $\mathcal{R}_{p \in \{p \mid p = \langle v, v \rangle \land C(\langle v, v \rangle)\}} \mathcal{F}(p)]$ 2392 From [34] and [13], we have (35) $\mathcal{S}_{\text{pull}+}^{k+1}(v) = \mathcal{R}[$ 2393 2394 $\mathcal{R}_{u \in \operatorname{preds}(v)} \mathcal{P}\left(\mathcal{R}_{p \in \{p \mid p \in \operatorname{Paths}(u) \land C(p) \land \operatorname{length}(p) < k\}} \mathcal{F}(p), \langle u, v \rangle\right),$ 2395 $\mathcal{R}_{p \in \{p \mid p = \langle v, v \rangle \land C(\langle v, v \rangle)\}} \mathcal{F}(p) \end{bmatrix}$ 2396 In the case that the size of the set of paths is more than one, from [8], and 2397 in the case that the set of paths is singleton, from $\mathcal{R}_{\{v\}} = v$ and [9], and 2398 in the case that the set of paths is empty, from $\mathcal{R}_{\emptyset} = \bot$ and [7], 2399 (36) $\mathcal{P}\left(\mathcal{R}_{p \in \{p \mid p \in \text{Paths}(u) \land C(p) \land \text{length}(p) < k\}}\mathcal{F}(p), \langle u, v \rangle\right) =$ 2400 2401

Anon.

2402	$\mathcal{R}_{p,\epsilon} \in \{p \mid p \in Path_{\epsilon}(u) \land \mathcal{L}(p) \land langth_{\epsilon}(p) < k\} \mathcal{F}(p \cdot \langle u, v \rangle)$
2403	From [35] and [36] we have
2404	(37) $S^{k+1}(v) - \mathcal{R}[$
2405	$(37) \mathcal{O}_{\text{pull}+}(0) = \mathcal{N}_{\text{pull}+}(0) = $
2405	$\mathcal{K}_{u \in \text{preds}(v)} \ \mathcal{K}_{p \in \{p \mid p \in \text{Paths}(u) \land C(p) \land \text{length}(p) < k\}} \mathcal{F}(p \cdot \langle u, v \rangle),$
2400	$\mathcal{K}_{p \in \{p \mid p = \langle v, v \rangle \land C(\langle v, v \rangle)\}}\mathcal{F}(p)\}$
2407	that is
2408	(38) $\mathcal{S}_{\text{pull+}}^{\kappa+1}(v) = \mathcal{R}[$
2409	$\mathcal{R}_{u \in preds(v)} \ \mathcal{R}_{p' \in \{p' \mid p' = p \cdot \langle u, v \rangle \land p \in Paths(u) \land C(p) \land length(p) < k\}} \mathcal{F}(p'),$
2410	$\mathcal{R}_{p \in \{p \mid p = \langle v, v \rangle \land C(\langle v, v \rangle)\}} \mathcal{F}(p)]$
2411	From [25] and Lemma 9,
2412	$(39) \mathcal{S}_{m,0}^{k+1}(v) = \mathcal{R}[$
2413	$\mathcal{R}_{\text{respective}}(x) = \mathcal{R}_{\text{respective}}(x) + \mathcal{R}_{respecti$
2414	$\mathcal{R} = \{p \mid p = p : \langle u, v \rangle \land p \in \{p \mid p = p : \langle u, v \rangle \land p \in Paths(u) \land C(p : \langle u, v \rangle) \land length(p) < k\} \neq \{p \mid p = p : \langle u, v \rangle \land p \in Paths(u) \land C(p : \langle u, v \rangle) \land length(p) < k\} \neq \{p \mid p = p : \langle u, v \rangle \land p \in Paths(u) \land C(p : \langle u, v \rangle) \land length(p) < k\} \neq \{p \mid p = p : \langle u, v \rangle \land p \in Paths(u) \land C(p : \langle u, v \rangle) \land length(p) < k\} \neq \{p \mid p = p : \langle u, v \rangle \land p \in Paths(u) \land C(p : \langle u, v \rangle) \land length(p) < k\} \neq \{p \mid p = p : \langle u, v \rangle \land p \in Paths(u) \land C(p : \langle u, v \rangle) \land length(p) < k\} \neq \{p \mid p = p : \langle u, v \rangle \land p \in Paths(u) \land C(p : \langle u, v \rangle) \land length(p) < k\} \neq \{p \mid p = p : \langle u, v \rangle \land p \in Paths(u) \land C(p : \langle u, v \rangle) \land length(p) < k\} \neq \{p \mid p = p : \langle u, v \rangle \land p \in Paths(u) \land C(p : \langle u, v \rangle) \land Paths(p) < k\}$
2415	From [30] [2] and [3] we have
2416	(40) $S^{k+1}(x) = \mathcal{O}[$
2417	$(40) \ \mathcal{S}_{\text{pull}+}(b) = \mathcal{K}[$
2418	$\mathcal{R}_{p'} \in \{p' \mid \exists u. \ u \in preds(v) \land p' = p \cdot \langle u, v \rangle \land p \in Paths(u) \land C(p \cdot \langle u, v \rangle) \land length(p) < k\} \mathcal{F}(p'),$
2419	$\mathcal{R}_{p \in \{p \mid p = \langle v, v \rangle \land C(\langle v, v \rangle)\}} \mathcal{F}(p)]$
2420	that is
2421	(41) $\mathcal{S}_{\text{pull}+}^{k+1}(v) = \mathcal{R}[$
2422	$\mathcal{R}_{p' \in \{p' \mid p \in \text{Paths}(p) \land C(p') \land 0 < \text{length}(p') < k+1\}} \mathcal{F}(p'),$
2422	$\mathcal{R}_{p \in \{p \mid p = \langle p, v \rangle \land C(\langle p, v \rangle)\}} \mathcal{F}(p)]$
2423	that is
2424	$\mathcal{S}^{k+1}_{k+1}(v) = \mathcal{R}_{p' \in \{p' \mid p \in Paths(p) \land C(p') \land length(p') < k+1\}} \mathcal{F}(p')$
2425	$pull + \langle y \rangle = p \langle p \rangle + p \langle u u u u \langle v \rangle + o \langle p \rangle + (u u u u \langle v \rangle) + o \langle p \rangle + (u u \langle v \rangle) + o \langle p \rangle + (u u \langle v \rangle) + o \langle p \rangle + (u u \langle v \rangle) + o \langle p \rangle + (u u \langle v \rangle) + o \langle p \rangle + (u u \langle v \rangle) + o \langle p \rangle + (u u \langle v \rangle) + o \langle p \rangle + (u u \langle v \rangle) + o \langle p \rangle + (u u \langle v \rangle) + o \langle p \rangle + (u u \langle v \rangle) + o \langle p \rangle + (u u \langle v \rangle) + o \langle p \rangle + (u u \langle v \rangle) + o \langle p \rangle + (u u \langle v \rangle) + o \langle p \rangle + (u u \langle v \rangle) + o \langle p \rangle + (u u \langle v \rangle) + o \langle p \rangle + (u u \langle v \rangle) + o \langle p \rangle + (u u \langle v \rangle) + o \langle p \rangle + (u u \langle v \rangle) + o \langle p \rangle + (u u \langle v \rangle) + o \langle p \rangle + (u \langle v \rangle) + (u \langle $
2426	
2427	LEMMA 9.
2428	$\forall p, v. \text{ let } u \coloneqq \text{tail}(p) \text{ in } C(p) \leftrightarrow C(p \cdot \langle u, v \rangle)$
2429	Proof
2430	We consider the two cases:
2431	Cose:
2432	(1) C(p) = (hard(p) = q)
2433	(1) $C(p) = (head(p) = s)$
2434	Straignitionward by $h_{res} f(r_{res}(u, r)) = 0$
2435	$neaa(p) = s \leftrightarrow neaa(p \cdot \langle u, v \rangle) = s$
2436	(2) $U(p) = 1$ rue
2437	Straightforward by
2438	True \leftrightarrow True
2439	
2440	
2441	
2442	
2443	
2444	
2445	
2113	
2440	
2447	
2448	
1001	
2449	

```
THEOREM 23 (CORRECTNESS OF PULL (NON-IDEMPOTENT REDUCTION)).
2453
                 For all \mathcal{R}, \mathcal{F}, \mathcal{I}, \mathcal{P}, k \geq 1 and s,
2454
                 let C(p) = (\text{head}(p) = s), and
2455
                 there is no cycle that contains s,
2456
                 if the conditions \mathbb{C}_1 - \mathbb{C}_8 hold,
2457
                 S_{\text{pull}-}^k(v) = Spec^k(v)
2458
2459
           Proof.
2460
           We assume that
2461
                  (1) \forall n. \mathcal{R}(n, \perp) = n
2462
                  (2) \forall n, n'. \mathcal{R}(n, n') = \mathcal{R}(n', n)
2463
                  (3) \forall n, n', n''. \mathcal{R}(\mathcal{R}(n, n'), n'') = \mathcal{R}(n, \mathcal{R}(n', n''))
2464
                  (4) \forall v \in V. C(\langle v, v \rangle) \rightarrow I(v) = \mathcal{F}(\langle v, v \rangle)
2465
                  (5) \forall v \in \mathsf{V}. \neg C(\langle v, v \rangle) \rightarrow I(v) = \bot
2466
                  (6) \forall e. \mathcal{P}(\perp, e) = \perp
2467
                  (7) \forall p_1, p_2 \in \mathsf{P}, v \in \mathsf{V}.
2468
2469
                             tail(p_1) = tail(p_2) \rightarrow
2470
                             let u \coloneqq tail(p_1) in
                             \mathcal{P}\left[\mathcal{R}(\mathcal{F}(p_1), \mathcal{F}(p_2)), \langle u, v \rangle\right] =
2471
                             \mathcal{R}\left[\mathcal{F}(p_1 \cdot \langle u, v \rangle), \mathcal{F}(p_2 \cdot \langle u, v \rangle)\right]
2472
                   (8) \forall p, e. \mathcal{P}(\mathcal{F}(p), e) = \mathcal{F}(p \cdot e)
2473
2474
                  (9) C(p) = (head(p) = s)
2475
                  (10) There is no cycle that contains s.
2476
           Form Def. 6, we have
2477
                  Spec^{k}(v) = \mathcal{R}_{p \in \{p \mid p \in Paths(v) \land C(p) \land length(p) < k\}} \mathcal{F}(p)
2478
2479
           Proof by induction on k:
2480
           Base Case:
2481
                  k = 1
2482
                   We should show that
2483
                        \mathcal{S}^{1}_{\mathsf{pull}-}(v) = \mathcal{R}_{p \in \{p \mid p \in \mathsf{Paths}(v) \land C(p) \land \mathsf{length}(p) < 1\}} \mathcal{F}(p)
2484
                   that is
2485
                        \mathcal{S}^1_{\mathsf{pull}-}(v) = \mathcal{R}_{p \in \{p \mid p = \langle v, v \rangle \land C(p)\}} \mathcal{F}(p)
2486
                  We consider two cases:
2487
                   Case:
2488
                        (11) C(\langle v, v \rangle)
2489
                        We should show that
2490
                             \mathcal{S}^{1}_{\mathsf{pull}-}(v) = \mathcal{R}_{\{\langle v, v \rangle\}} \mathcal{F}(p)
2491
                        that is
2492
                             \mathcal{S}^{1}_{\mathsf{pull}-}(v) = \mathcal{F}(\langle v, v \rangle)
2493
                        that is straightforward from Def. 2, [4] and [11].
2494
                   Case:
2495
                        (12) \neg C(\langle v, v \rangle)
2496
                        We should show that
2497
                             \mathcal{S}^1_{\mathsf{pull}-}(v) = \mathop{\mathcal{R}}_{\emptyset} \mathcal{F}(p)
2498
2499
```

Anon.

2500	that is
2501	$S^{1} = (v) = 1$
2502	that is straightforward from Def 2 [5] and [12]
2503	that is straightforward from Del. 2, [5] and [12].
2504	Inductive Case:
2505	(13) $k > 1$
2506	The induction hypothesis is:
2507	(14) $\mathcal{S}^{k'}$ $(v) = \mathcal{R}_{p \in \{p \mid p \in \text{Paths}(v) \land C(p) \land \text{length}(p) < k'\}} \mathcal{F}(p)$ for all v and $k' < k$
2508	We should show that $(0, 1) = (1, 1) = (1, 1)$
2509	$S^{k+1}(v) = \mathcal{R}_{p \in \{p \mid p \in \text{Paths}(v) \land C(p) \land \text{length}(p) < k+1\}}\mathcal{F}(p)$
2510	We consider two cases:
2511	Case:
2512	(15) $v = s$
2513	By Lemma 11 on [9] and [4], [5], and [10],
2514	(16) $S_{mull}^{k+1}(s) = I(s)$
2515	Bv [4] and [9].
2516	(17) $I(s) = \mathcal{F}(\langle s, s \rangle)$
2517	From [16] and [17],
2518	(18) $S_{\text{pull}}^{k+1}(s) = \mathcal{F}(\langle s, s \rangle)$
2519	From [9] and [10],
2520	(19) $\mathcal{R}_{p \in \{p \mid p \in \text{Paths}(s) \land C(p) \land \text{length}(p) \le k+1\}} \mathcal{F}(p) =$
2521	$\mathcal{R}_{p \in \{p \mid p \in \text{Paths} \land \text{tail}(p) = \land \text{head}(p) = \land he$
2522	$\mathcal{R}_{p \in \{\langle s, s \rangle\}} \mathcal{F}(p) =$
2523	$\mathcal{F}(\langle s, s \rangle)$
2525	From [18] and [19],
2526	$\mathcal{S}^{k+1}_{pull-}(v) = \mathcal{R}_{p \in \{p \mid p \in Paths(v) \land C(p) \land length(p) < k+1\}}\mathcal{F}(p)$
2527	
2528	Case:
2529	$(20) v \neq s$
2530	From Def. 2, we consider two sub-cases:
2531	Sub-case:
2532	(21) $\operatorname{CPreds}^{\kappa}(v) \neq \emptyset$
2533	(22) $\mathcal{S}_{\text{pull}-}^{k+1}(v) = \mathcal{R}_{u \in \text{preds}(v)} \mathcal{P}\left(\mathcal{S}_{\text{pull}-}^{k}(u), \langle u, v \rangle\right)$
2534	From [22] and [14]
2535	(23) $S_{\text{pull}}^{k+1}(v) = \mathcal{R}_{u \in \text{preds}(v)} \left[\mathcal{P} \left(\mathcal{R}_{p \in \{p \mid p \in \text{Paths}(u) \land C(p) \land \text{length}(p) < k\}} \mathcal{F}(p), \langle u, v \rangle \right) \right]$
2536	In the case that the size of the set of paths is more than one, from [7], and
2537	in the case that the set of paths is singleton, from $\mathcal{R}_{\{v\}} = v$ and [8], and
2538	in the case that the set of paths is empty, from $\mathcal{R}_{\emptyset} = \bot$ and [6],
2539	we have
2540	(24) $\mathcal{P}\left(\mathcal{R}_{p \in \{p \mid p \in Paths(u) \land C(p) \land length(p) < k\}}\mathcal{F}(p), \langle u, v \rangle\right) =$
2541	$\mathcal{R}_{p \in \{p \mid p \in Paths(u) \land C(p) \land length(p) < k\}} \mathcal{F}(p \cdot \langle u, v angle)$
2542	After substituting [24] in [23]
2543	(25) $\mathcal{S}_{\text{pull}-}^{k+1}(v) = \mathcal{R}_{u \in \text{preds}(v)} \left[\mathcal{R}_{p \in \{p \mid p \in \text{Paths}(u) \land C(p) \land \text{length}(p) < k\}} \mathcal{F}(p \cdot \langle u, v \rangle) \right]$
2544	that is
2545	(26) $S_{\text{pull}-}^{k+1}(v) = \mathcal{R}_{p \in \{p \mid \exists u. \ p \in \text{Paths}(u) \land u \in \text{preds}(v) \land C(p) \land \text{length}(p) < k\}} \mathcal{F}(p \cdot \langle u, v \rangle)$
2547	From [26] and Lemma 9
2548	
2010	

(27) $S_{\text{pull}-}^{k+1}(v) = \mathcal{R}_{p \in \{p \mid \exists u. \ p \in \text{Paths}(u) \land u \in \text{preds}(v) \land C(p \cdot \langle u, v \rangle) \land \text{length}(p) < k\}} \mathcal{F}(p \cdot \langle u, v \rangle)$ 2549 2550 that is (28) $\mathcal{S}_{\text{pull}-}^{k+1}(v) = \mathcal{R}_{p' \in \{p' \mid p' \in \text{Paths}(v) \land C(p') \land 0 < \text{length}(p') < k+1\}} \mathcal{F}(p')$ 2551 2552 From [28] and [20] 2553 $\mathcal{S}_{\text{push+}}^{k+1}(v) = \mathcal{R}_{p \in \{p \mid p \in \text{Paths}(v) \land C(p) \land \text{length}(p) < k+1\}} \mathcal{F}(p),$ 2554 2555 Sub-case: 2556 (29) $\operatorname{CPreds}^k(v) = \emptyset$ 2557 (30) $\mathcal{S}_{\text{pull}-}^{k+1}(v) = \mathcal{S}_{\text{pull}-}^{k}(v)$ 2558 (31) $\operatorname{CPreds}^{k}(v) = \left\{ u \mid u \in \operatorname{preds}(v) \land \mathcal{S}_{\operatorname{pull}}^{k}(u) \neq \mathcal{S}_{\operatorname{pull}}^{k-1}(u) \right\}$ 2559 From [30] and [14], 2560 (32) $S_{\text{pull}-}^{k+1}(v) = \mathcal{R}_{p \in \{p \mid p \in \text{Paths}(v) \land C(p) \land \text{length}(p) < k\}}\mathcal{F}(p)$ 2561 2562 From [32] and [20], $(33) \ \mathcal{S}_{\mathsf{pull}-}^{k+1}(v) = \mathcal{R}_{p \in \{p \mid \exists u. \ p \in \mathsf{Paths}(u) \land u \in \mathsf{preds}(v) \land C(p \cdot \langle u, v \rangle) \land 0 < \mathsf{length}(p \cdot \langle u, v \rangle) < k\}}\mathcal{F}(p)$ 2563 2564 From [33] and Lemma 9, (34) $\tilde{\mathcal{S}}_{\text{pull}-}^{k+1}(v) = \mathcal{R}_{p \in \{p \mid \exists u. \ p \in \text{Paths}(u) \land u \in \text{preds}(v) \land C(p) \land 0 < \text{length}(p \cdot \langle u, v \rangle) < k\}} \mathcal{F}(p)$ 2565 2566 From [34], [2] and [3] 2567 (35) $S_{\text{pull}-}^{k+1}(v) = \mathcal{R}_{u \in \text{preds}(v)} \mathcal{R}_{p' \in \{p' \mid \exists u. \ p' \in \text{Paths}(u) \land C(p') \land \text{length}(p') < k-1\}} \mathcal{F}(p' \cdot \langle u, v \rangle)$ 2568 In the case that the size of the set of paths is more than one, from [7], and 2569 in the case that the set of paths is singleton, from $\mathcal{R}_{\{v\}} = v$ and [8], and 2570 in the case that the set of paths is empty, from $\mathcal{R}_{\emptyset} = \bot$ and [6], 2571 (36) $\mathcal{P}\left(\mathcal{R}_{p \in \{p \mid p \in \text{Paths}(u) \land C(p) \land \text{length}(p) < k-1\}}\mathcal{F}(p), \langle u, v \rangle\right) =$ 2572 $\mathcal{R}_{p \in \{p \mid p \in \text{Paths}(u) \land C(p) \land \text{length}(p) < k-1\}} \mathcal{F}(p \cdot \langle u, v \rangle)$ 2573 From [35] and [36], we have 2574 $(37) \ \mathcal{S}_{\text{pull}-}^{k+1}(v) = \mathcal{R}_{u \in \text{preds}(v)} \ \mathcal{P}\left(\mathcal{R}_{p \in \{p \mid p \in \text{Paths}(u) \land C(p) \land \text{length}(p) < k-1\}}\mathcal{F}(p), \ \langle u, v \rangle\right)$ 2575 From [37] and [14], we have 2576 (38) $S_{\text{pull}-}^{k+1}(v) = \mathcal{R}_{u \in \text{preds}(v)} \mathcal{P}\left(S_{\text{pull}-}^{k-1}(u), \langle u, v \rangle\right)$ 2577 From [29] and [31], we have 2578 (39) For all $u \in \operatorname{preds}(v)$: $S_{\operatorname{pull}}^k(u) = S_{\operatorname{pull}}^{k-1}(u)$ 2579 2580 From [38] and [39], we have 2581 (40) $\mathcal{S}_{\text{pull}-}^{k+1}(v) = \mathcal{R}_{u \in \text{preds}(v)} \mathcal{P}\left(\mathcal{S}_{\text{pull}-}^{k}(u), \langle u, v \rangle\right)$ 2582 From [40] and [14], we have 2583 (41) $\mathcal{S}_{\text{pull}-}^{k+1}(v) = \mathcal{R}_{u \in \text{preds}(v)} \mathcal{P}\left(\mathcal{R}_{p \in \{p \mid p \in \text{Paths}(u) \land C(p) \land \text{length}(p) < k\}}\mathcal{F}(p), \langle u, v \rangle\right)$ 2584 In the case that the size of the set of paths is more than one, from [7], and 2585 in the case that the set of paths is singleton, from $\mathcal{R}_{\{v\}} = v$ and [8], and 2586 in the case that the set of paths is empty, from $\mathcal{R}_{\emptyset} = \bot$ and [6], 2587 (42) $\mathcal{P}\left(\mathcal{R}_{p \in \{p \mid p \in \mathsf{Paths}(u) \land C(p) \land \mathsf{length}(p) < k\}}\mathcal{F}(p), \langle u, v \rangle\right) =$ 2588 $\mathcal{R}_{p \in \{p \mid p \in \mathsf{Paths}(u) \land C(p) \land \mathsf{length}(p) < k\}} \mathcal{F}(p \cdot \langle u, v \rangle)$ 2589 From [41] and [42], we have 2590 (43) $S_{\text{pull}-}^{k+1}(v) = \mathcal{R}_{u \in \text{preds}(v)} \mathcal{R}_{p \in \{p \mid p \in \text{Paths}(u) \land C(p) \land \text{length}(p) < k\}} \mathcal{F}(p \cdot \langle u, v \rangle)$ 2591 that is 2592 (44) $S_{\text{pull}-}^{k+1}(v) = \mathcal{R}_{u \in \text{preds}(v)} \mathcal{R}_{p' \in \{p' \mid p'=p \cdot \langle u, v \rangle \land p \in \text{Paths}(u) \land C(p) \land \text{length}(p) < k\}} \mathcal{F}(p')$ 2593 From [31] and Lemma 9, 2594 (45) $S_{\text{null}-}^{k+1}(v) = \mathcal{R}_{u \in \text{preds}(v)} \mathcal{R}_{p' \in \{p' \mid p' = p \cdot \langle u, v \rangle \land p \in \text{Paths}(u) \land C(p \cdot \langle u, v \rangle) \land \text{length}(p) < k\}} \mathcal{F}(p')$ 2595 2596 2597

2598	From [45], [2] and [3], we have
2599	(46) $S_{\text{null}-}^{k+1}(v) = \mathcal{R}_{p' \in \{p' \mid \exists u. \ u \in \text{preds}(v) \land p' = p \cdot \langle u, v \rangle \land p \in \text{Paths}(u) \land C(p \cdot \langle u, v \rangle) \land \text{length}(p) < k\}} \mathcal{F}(p')$
2600	that is
2601	(47) $\mathcal{S}_{m,0}^{k+1}(v) = \mathcal{R}_{p' \in \{p' \mid p \in \text{Paths}(v) \land C(p') \land 0 \le \text{length}(p') \le k+1\}} \mathcal{F}(p')$
2602	From [47] and [20], we have
2603	$S^{k+1}_{(v)}(v) = \mathcal{R}_{v' \in \{p' \mid p \in Path_{v(v)}\} \land C(p') \land length_{(p') < k+1} \lor \mathcal{F}(p')$
2604	$pull = (0)$ $p \in \{p \mid p \in aus(0) \land C(p) \land ungun(p) < x+1\}$ (p)
2605	
2606	Lemma 10.
2607	For all $\mathcal{R}, \mathcal{F}, C, I, \mathcal{P}, k \geq 1$ and s, v, where
2608	(1) C(p) = (head(p) = s),
2609	$(2) \forall v \in \mathbb{V}. \ C(\langle v, v \rangle) \to I(v) = \mathcal{F}(\langle v, v \rangle)$
2610	$(3) \forall v \in \mathbb{V}. \neg C(\langle v, v \rangle) \to I(v) = \bot$
2611	$if \mathcal{S}_{\text{pull}-}^{\kappa}(v) \neq \mathcal{S}_{\text{pull}-}^{\kappa-1}(v),$
2612	then v is reachable from s.
2613	Proof
2614	Immediate from induction on k and case analysis on branches of Def. 2
2615	The base case is from [1] [2] and [3]
2616	
2617	
2618	Lемма 11.
2619	For all $\mathcal{R}, \mathcal{F}, \mathcal{C}, \mathcal{I}, \mathcal{P}, k \geq 1$ and s, v, where
2620	(1) C(p) = (head(p) = s),
2621	$(2) \forall v \in \mathbb{V}. \ C(\langle v, v \rangle) \to I(v) = \mathcal{F}(\langle v, v \rangle)$
2622	$(3) \forall v \in V. \neg C(\langle v, v \rangle) \to \mathcal{I}(v) = \bot$
2623	there is no cycle that contains s,
2624	then $S_{\text{pull}-}^{\kappa}(s) = I(s)$.
2625	Proof
2626	Immediate from induction on k and case analysis on branches of Def. 2.
2627	The second branch is refuted by Lemma 10 and the assumption of acyclicity for s.
2628	r
2629	
2630	
2631	
2632	
2633	
2634	
2635	
2030	
2037	
2030	
2039	
2641	
2642	
2643	
2644	
2645	
2646	
2010	

```
THEOREM 24 (CORRECTNESS OF PUSH (IDEMPOTENT REDUCTION)).
2649
                 For all \mathcal{R}, \mathcal{F}, C, I, \mathcal{P}, and k \geq 1, if the conditions \mathbb{C}_1 - \mathbb{C}_9 hold,
2650
                 S_{\text{nush+}}^k(v) = Spec^k(v)
2651
2652
           Proof.
2653
           We assume that
2654
                  (1) \forall n. \mathcal{R}(n, \perp) = n
2655
                  (2) \forall n, n', \mathcal{R}(n, n') = \mathcal{R}(n', n)
2656
                  (3) \forall n, n', n''. \mathcal{R}(\mathcal{R}(n, n'), n'') = \mathcal{R}(n, \mathcal{R}(n', n''))
2657
                  (4) \forall n. \mathcal{R}(n, n) = n
2658
                  (5) \forall v \in \mathsf{V}. \ C(\langle v, v \rangle) \to \mathcal{I}(v) = \mathcal{F}(\langle v, v \rangle)
2659
                  (6) \forall v \in V. \neg C(\langle v, v \rangle) \rightarrow I(v) = \bot
2660
                  (7) \forall e. \mathcal{P}(\perp, e) = \perp
2661
2662
                  (8) \forall p_1, p_2 \in \mathsf{P}, v \in \mathsf{V}.
2663
                              tail(p_1) = tail(p_2) \rightarrow
2664
                              let u := tail(p_1) in
2665
                              \mathcal{P}\left[\mathcal{R}(\mathcal{F}(p_1), \mathcal{F}(p_2)), \langle u, v \rangle\right] =
2666
                              \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)
2667
2668
           Form Def. 6, we have
                  Spec^{k}(v) = \mathcal{R}_{p \in \{p \mid p \in \text{Paths}(v) \land C(p) \land \text{length}(p) < k\}} \mathcal{F}(p)
2669
2670
2671
           Proof by induction on k:
2672
           Base Case:
                  We should show that
2673
                        S^{1}_{\mathsf{push+}}(v) = \mathcal{R}_{p \in \{p \mid p \in \mathsf{Paths}(v) \land C(p) \land \mathsf{length}(p) < 1\}} \mathcal{F}(p)
2674
2675
                  that is
                        \mathcal{S}^{1}_{\mathsf{push}+}(v) = \mathcal{R}_{p \in \{p \mid p = \langle v, v \rangle \land C(p)\}} \mathcal{F}(p)
2676
2677
                  We consider two cases:
2678
                   Case:
2679
                        (10) C(\langle v, v \rangle)
2680
                        We should show that
2681
                              \mathcal{S}^{1}_{\mathsf{push+}}(v) = \mathcal{R}_{\{\langle v, v \rangle\}}\mathcal{F}(p)
2682
                        that is
2683
                              \mathcal{S}^{1}_{\mathsf{push+}}(v) = \mathcal{F}(\langle v, v \rangle)
2684
                        that is straightforward from Def. 3, [5] and [10].
2685
                   Case:
2686
                        (11) \neg C(\langle v, v \rangle)
2687
                        We should show that
2688
                              \mathcal{S}^{1}_{\mathsf{push+}}(v) = \mathop{\mathcal{R}}_{\emptyset} \mathcal{F}(p)
2689
                        that is
2690
                              S^1_{\text{push+}}(v) = \bot
2691
                        that is straightforward from Def. 3, [6] and [11].
2692
2693
           Inductive Case:
2694
2695
```

Anon.

2696 The induction hypothesis is: (12) $S_{\text{push+}}^k(v) = \mathcal{R}_{p \in \{p \mid p \in \text{Paths}(v) \land C(p) \land \text{length}(p) < k\}}\mathcal{F}(p),$ 2697 k > 12698 We should show that $S_{\text{nush+}}^{k+1}(v) = \mathcal{R}_{p \in \{p \mid p \in \text{Paths}(v) \land C(p) \land \text{length}(p) < k+1\}} \mathcal{F}(p)$ 2699 2700 From Def. 3, we have that (13) $\mathcal{S}_{\text{push+}}^{k+1}(v) = S_n$ 2701 2702 (14) $\{u_1, ..., u_n\} = u \in \left\{ u \mid u \in \operatorname{preds}(v) \land \mathcal{S}_{\operatorname{push}+}^k(u) \neq \mathcal{S}_{\operatorname{push}+}^{k-1}(u) \right\}$ 2703 (15) $S_0 = \mathcal{S}_{\text{push}+}^k(v)$ 2704 (16) $S_{i+1} = \mathcal{R}\left(S_i, \mathcal{P}(\mathcal{S}_{push+}^k(u_i), \langle u_i, v \rangle)\right)$ 2705 2706 From [13]-[16], and [2] and [3], we have 2707 (17) $S_{\text{push+}}^{k+1}(v) = \mathcal{R}\left[\mathcal{R}_{u \in \{u \mid u \in \text{preds}(v) \land S_{\text{push+}}^{k}(u) \neq S_{\text{push+}}^{k-1}(u)\}}\left[\mathcal{P}(S_{\text{push+}}^{k}(u), \langle u, v \rangle)\right], S_{\text{push+}}^{k}(v)\right]$ From Lemma 12, and [2], [3], and [4], we have 2708 2709 2710 (18) $\mathcal{S}_{\text{push+}}^{k}(v) = \mathcal{R}\left(\mathcal{S}_{\text{push+}}^{k}(v), \mathcal{R}_{u \in \{u \mid u \in \text{preds}(v) \land \mathcal{S}_{\text{push+}}^{k}(u) = \mathcal{S}_{\text{push+}}^{k-1}(u)\}}\mathcal{P}(\mathcal{S}_{\text{push+}}^{k}(u), \langle u, v \rangle)\right)$ 2711 After substituting [18] in [17] 2712 (19) $S_{\text{push+}}^{k+1}(v) = \mathcal{R}[\mathcal{R}_{u \in \{u \mid u \in \text{preds}(v) \land S_{\text{push+}}^{k}(u) \neq S_{\text{nush+}}^{k-1}(u)\}} \Big[\mathcal{P}(S_{\text{push+}}^{k}(u), \langle u, v \rangle) \Big],$ 2713 $\mathcal{R}(\mathcal{S}_{\mathsf{push+}}^{k}(v), \mathcal{R}_{u \in \{u \mid u \in \mathsf{preds}(v) \land \mathcal{S}_{\mathsf{push+}}^{k}(u) = \mathcal{S}_{\mathsf{push+}}^{k-1}(u)\}}\left[\mathcal{P}(\mathcal{S}_{\mathsf{push+}}^{k}(u), \langle u, v \rangle)\right])\right]$ 2714 2715 From [19], [2] and [3] 2716 (20) $\mathcal{S}_{\text{push+}}^{k+1}(v) = \mathcal{R} \left[\mathcal{R}_{u \in \text{preds}(v)} \left[\mathcal{P}(\mathcal{S}_{\text{push+}}^k(u), \langle u, v \rangle) \right], \mathcal{S}_{\text{push+}}^k(v) \right]$ 2717 2718 From [20] and [12] 2719 (21) $\mathcal{S}_{\text{push+}}^{k+1}(v) = \mathcal{R}[$ 2720 $\mathcal{R}_{u \in \text{preds}(v)} \left[\mathcal{P} \left(\mathcal{R}_{p \in \{p \mid p \in \text{Paths}(u) \land C(p) \land \text{length}(p) < k \}} \mathcal{F}(p), \langle u, v \rangle \right) \right],$ 2721 $\mathcal{S}^k_{\mathrm{push+}}(v)]$ 2722 In the case that the size of the set of paths is more than one, from [8], and 2723 in the case that the set of paths is singleton, from $\mathcal{R}_{\{v\}} = v$ and [9], and 2724 in the case that the set of paths is empty, from $\mathcal{R}_{\emptyset} = \bot$ and [7], 2725 we have 2726 (22) $\mathcal{P}(\mathcal{R}_{p \in \{p \mid p \in \text{Paths}(u) \land C(p) \land \text{length}(p) < k\}}\mathcal{F}(p), \langle u, v \rangle) =$ 2727 $\mathcal{R}_{p \in \{p \mid p \in \mathsf{Paths}(u) \land C(p) \land \mathsf{length}(p) < k\}} \mathcal{F}(p \cdot \langle u, v \rangle)$ 2728 After substituting [22] in [21] 2729 (23) $\mathcal{S}^{k+1}_{\text{push}+}(v) = \mathcal{R}[$ 2730 $\mathcal{R}_{u \in \text{preds}(v)} \left[\mathcal{R}_{p \in \{p \mid p \in \text{Paths}(u) \land C(p) \land \text{length}(p) < k\}} \mathcal{F}(p \cdot \langle u, v \rangle) \right],$ 2731 $\mathcal{S}^k_{\text{nush}}(v)$] 2732 2733 that is (24) $\mathcal{S}^{k+1}_{\text{push+}}(v) = \mathcal{R}[$ 2734 2735 $\mathcal{R}_{p \in \{p \mid \exists u. \ p \in \mathsf{Paths}(u) \land u \in \mathsf{preds}(v) \land C(p) \land \mathsf{length}(p) < k\}} \mathcal{F}(p \cdot \langle u, v \rangle),$ 2736 $\mathcal{S}^k_{\mathsf{push+}}(v)]$ 2737 From [24] and Lemma 9 (25) $\mathcal{S}^{k+1}_{\text{push+}}(v) = \mathcal{R}[$ 2738 2739 $\begin{aligned} &\mathcal{R}_{p \in \{p \mid \exists u. \ p \in \mathsf{Paths}(u) \land u \in \mathsf{preds}(v) \land C(p \cdot \langle u, v \rangle) \land \mathsf{length}(p) < k\}} \mathcal{F}(p \cdot \langle u, v \rangle), \\ &\mathcal{S}_{\mathsf{push+}}^{k}(v) \end{bmatrix} \end{aligned}$ 2740 2741 that is 2742 (26) $\mathcal{S}_{\text{push}+}^{k+1}(v) = \mathcal{R}[$ 2743 2744

2745	$\mathcal{R}_{p' \in \{p' \mid p' \in Paths(v) \land C(p') \land length(p') < k+1\}} \mathcal{F}(p'),$
2746	$\mathcal{S}^k_{nuch}(v)]$
2747	From [26] and [12]
2748	(27) $\mathcal{S}_{\text{nucl}}^{k+1}(v) = \mathcal{R}[$
2749	$\mathcal{R}_{p \in \{p \mid p \in \text{Paths}(p) \land C(p) \land \text{length}(p) < k+1\}} \mathcal{F}(p),$
2750	$\mathcal{R}_{p \in \{p \mid p \in Paths(p) \land C(p) \land length(p) < k\}} \mathcal{F}(p)]$
2751	From [27] and [4]
2752	(28) $S_{\text{purp}+}^{k+1}(v) = \mathcal{R}_{p \in \{p \mid p \in \text{Paths}(v) \land C(p) \land \text{length}(p) < k+1\}} \mathcal{F}(p),$
2753	
2754	LEMMA 12. Γ
2755	For all $\mathcal{R}, \mathcal{F}, \mathcal{C}, \mathcal{I}, \mathcal{P}$, if the conditions $\mathbb{C}_1 - \mathbb{C}_{10}$ hold,
2756	$\forall v, u, \kappa.$
2757	$k \ge 1 \land u \in \operatorname{preds}(v) \land \mathcal{S}_{\operatorname{push}+}^{n}(u) = \mathcal{S}_{\operatorname{push}+}^{n-1}(u) \to$
2750	$\mathcal{S}_{\text{push+}}^{\kappa}(v) = \mathcal{R}(\mathcal{S}_{\text{push+}}^{\kappa}(v), \mathcal{P}(\mathcal{S}_{\text{push+}}^{\kappa}(u), \langle u, v \rangle))$
2759	Proof
2761	We assume that
2762	(1) $\forall n, \mathcal{R}(n, \perp) = n$
2763	(2) $\forall n, n', \mathcal{R}(n, n') = \mathcal{R}(n', n)$
2764	(3) $\forall n, n', n'', \mathcal{R}(\mathcal{R}(n, n'), n'') = \mathcal{R}(n, \mathcal{R}(n', n''))$
2765	(4) $\forall n, \mathcal{R}(n, n) = n$
2766	(5) $\forall n \in V$ $C(\langle n n \rangle) \to I(n) = \mathcal{F}(\langle n n \rangle)$
2767	(c) $\forall v \in V$ $\neg C(\langle v, v \rangle) \rightarrow I(v) = 1$
2768	(c) $\forall e \in V$. $\Theta((e, e)) \to 2(e) = \pm$ (7) $\forall e = \Phi(+e) = \pm$
2769	$(7) \forall c, f (\pm, c) = \pm $ $(8) \forall c, f (\pm, c) = f (\pm, c) $
2770	(6) $\forall p_1, p_2 \in \mathbb{N}, \ 0 \in \mathbb{V}.$
2771	$\operatorname{tail}(p_1) \to \operatorname{tail}(p_2) \to$
2772	$let u := tail(p_1) in$
2773	$\mathcal{P}\left[\mathcal{R}(\mathcal{F}(p_1) \mid \mathcal{F}(p_2)) \mid \langle u \mid v \rangle\right] =$
2774	$\mathcal{R}\left[\mathcal{F}(p_1, \langle u, v \rangle), \mathcal{F}(p_2, \langle u, v \rangle)\right]$
2775	(9) $\forall p, e, \mathcal{P}(\mathcal{F}(p), e) = \mathcal{F}(p \cdot e)$
2776	
2777	Proof by induction on <i>k</i> :
2778	Base Case:
2779	(10) $k = 1$
2780	We assume that
2781	(11) $u \in \operatorname{preds}(v)$
2782	(12) $S_{\text{puch}+}^1(u) = S_{\text{puch}+}^0(u)$
2784	From Def. 3 on [12]
2785	(13) $S^1_{\text{nucl}}(u) = \bot$
2786	We need to show that
2787	(14) $\mathcal{S}^{1}_{\text{nuch}+}(v) = \mathcal{R}(\mathcal{S}^{1}_{\text{nuch}+}(v), \mathcal{P}(\mathcal{S}^{1}_{\text{nuch}+}(u), \langle u, v \rangle))$
2788	From [13] and [7], we need to show that
2789	(15) $\mathcal{S}^1_{\text{nuch}\pm}(v) = \mathcal{R}(\mathcal{S}^1_{\text{nuch}\pm}(v), \perp)$
2790	that is immediate from [1].
2791	
2792	Inductive Case:
2793	

2794 The induction hypothesis is 2795 (16) $\forall v, u$. $\begin{aligned} u \in \mathrm{preds}(v) \ \land \ \mathcal{S}_{\mathrm{push+}}^{k-1}(u) &= \mathcal{S}_{\mathrm{push+}}^{k-2}(u) \rightarrow \\ \mathcal{S}_{\mathrm{push+}}^{k-1}(v) &= \mathcal{R}(\mathcal{S}_{\mathrm{push+}}^{k-1}(v), \mathcal{P}(\mathcal{S}_{\mathrm{push+}}^{k-1}(u), \langle u, v \rangle)) \end{aligned}$ 2796 2797 2798 We assume that 2799 (17) k > 12800 (18) $u \in \operatorname{preds}(v)$ 2801 (19) $\mathcal{S}_{\text{push+}}^k(u) = \mathcal{S}_{\text{push+}}^{k-1}(u)$ 2802 We show that 2803 $\mathcal{S}_{\text{nush}+}^{k}(v) = \mathcal{R}(\mathcal{S}_{\text{nush}+}^{k}(v), \mathcal{P}(\mathcal{S}_{\text{nush}+}^{k}(u), \langle u, v \rangle))$ 2804 2805 From Def. 3 on [17], we have that 2806 (20) $\mathcal{S}_{\text{push+}}^k(v) = S_n$ 2807 (21) $\{u_1, ..., u_n\} = u \in \left\{ u \mid u \in \text{preds}(v) \land S^{k-1}_{\text{push+}}(u) \neq S^{k-2}_{\text{push+}}(u) \right\}$ 2808 2809 (22) $S_0 = S_{\text{push}+}^{k-1}(v)$ 2810 (23) $S_{i+1} = \mathcal{R}\left(S_i, \mathcal{P}(\mathcal{S}_{\text{nush}+}^{k-1}(u_i), \langle u_i, v \rangle)\right)$ 2811 2812 From [20]-[23], and [2] and [3], we have $(24) \ \mathcal{S}_{\mathsf{push+}}^{k}(v) = \mathcal{R}\left[\mathcal{R}_{u \in \{u \mid u \in \mathsf{preds}(v) \land \mathcal{S}_{\mathsf{push+}}^{k-2}(u) \neq \mathcal{S}_{\mathsf{nush+}}^{k-1}(u)\}}\left[\mathcal{P}(\mathcal{S}_{\mathsf{push+}}^{k-1}(u), \langle u, v \rangle)\right], \ \mathcal{S}_{\mathsf{push+}}^{k-1}(v)\right]$ 2813 2814 We consider two cases: 2815 Case: 2816 (25) $\mathcal{S}^{k-2}_{\text{push+}}(u) \neq \mathcal{S}^{k-1}_{\text{push+}}(u)$ 2817 From [24], [25], and [4] we have 2818 (26) $\mathcal{S}_{\text{push+}}^{k}(v) = \mathcal{R}\left(\mathcal{S}_{\text{push+}}^{k}(v), \mathcal{P}(\mathcal{S}_{\text{push+}}^{k-1}(u), \langle u, v \rangle)\right)$ 2819 2820 From [26] and [19] $\mathcal{S}_{\mathsf{push+}}^{k}(v) = \mathcal{R}\left(\mathcal{S}_{\mathsf{push+}}^{k}(v), \mathcal{P}(\mathcal{S}_{\mathsf{push+}}^{k}(u), \langle u, v \rangle)\right)$ 2821 2822 Case: 2823 (27) $\mathcal{S}_{\text{push+}}^{k-2}(u) = \mathcal{S}_{\text{push+}}^{k-1}(u)$ 2824 From [24] and [4] we have 2825 (28) $\mathcal{S}_{\text{push+}}^{k}(v) = \mathcal{R}\left(\mathcal{S}_{\text{push+}}^{k}(v), \mathcal{S}_{\text{push+}}^{k-1}(v)\right)$ 2826 2827 From [27] and [19], we have (29) $S_{\text{push}+}^{k-1}(u) = S_{\text{push}+}^{k-2}(u)$ 2828 2829 From [16] on [18] and [29] and then [19], we have 2830 (30) $\mathcal{S}_{\text{push+}}^{k-1}(v) = \mathcal{R}(\mathcal{S}_{\text{push+}}^{k-1}(v), \mathcal{P}(\mathcal{S}_{\text{push+}}^{k}(u), \langle u, v \rangle))$ 2831 From [28], [30] and [2], we have 2832 $\mathcal{S}_{\text{push}+}^{k}(v) = \mathcal{R}(\mathcal{S}_{\text{push}+}^{k}(v), \mathcal{P}(\mathcal{S}_{\text{push}+}^{k}(u), \langle u, v \rangle))$ 2833 2834 2835 2836 2837 2838 2839 2840 2841

4.4.4 Push, Non-idempotent 2843 2844 2845 We consider the two variants in turn. The first variant of push, non-idempotent was defined in Fig. 8, Def. 4. 2846 2847 THEOREM 25 (CORRECTNESS OF PUSH (NON-IDEMPOTENT REDUCTION) I). 2848 For all $\mathcal{R}, \mathcal{F}, I, \mathcal{P}, k \geq 1$, and s, 2849 let C(p) = (head(p) = s), 2850 *if the conditions* \mathbb{C}_1 - \mathbb{C}_8 *hold, and* 2851 s is not on any cycle, 2852 $S_{\text{push-}}^k(v) = Spec^k(v)$ 2853 Proof. 2854 We assume that 2855 (1) $\forall n. \mathcal{R}(n, \perp) = n$ 2856 (2) $\forall n, n'. \mathcal{R}(n, n') = \mathcal{R}(n', n)$ 2857 2858 (3) $\forall n, n', n''$. $\mathcal{R}(\mathcal{R}(n, n'), n'') = \mathcal{R}(n, \mathcal{R}(n', n''))$ 2859 (4) $\forall v \in V. C(\langle v, v \rangle) \rightarrow I(v) = \mathcal{F}(\langle v, v \rangle)$ 2860 (5) $\forall v \in V. \neg C(\langle v, v \rangle) \rightarrow I(v) = \bot$ 2861 (6) $\forall e. \mathcal{P}(\bot, e) = \bot$ 2862 (7) $\forall p_1, p_2 \in \mathsf{P}, v \in \mathsf{V}.$ 2863 $tail(p_1) = tail(p_2) \rightarrow$ 2864 let $u := tail(p_1)$ in 2865 $\mathcal{P}\left[\mathcal{R}(\mathcal{F}(p_1), \mathcal{F}(p_2)), \langle u, v \rangle\right] =$ 2866 $\mathcal{R}\left[\mathcal{F}(p_1 \cdot \langle u, v \rangle), \, \mathcal{F}(p_2 \cdot \langle u, v \rangle)\right]$ 2867 (8) $\forall p, e. \mathcal{P}(\mathcal{F}(p), e) = \mathcal{F}(p \cdot e)$ 2868 (9) C(p) = (head(p) = s)2869 (10) There is no cycle that contains s. 2870 Form Def. 6, we have 2871 $Spec^{k}(v) = \mathcal{R}_{p \in \{p \mid p \in \text{Paths}(v) \land C(p) \land \text{length}(p) < k\}} \mathcal{F}(p)$ 2872 2873 Proof by induction on *k*: 2874 Base Case: 2875 k = 12876 We should show that 2877 $S^{1}_{\text{push-}}(v) = \mathcal{R}_{p \in \{p \mid p \in \text{Paths}(v) \land C(p) \land \text{length}(p) < 1\}} \mathcal{F}(p)$ 2878 that is 2879 $\mathcal{S}^{1}_{\mathsf{push}-}(v) = \mathcal{R}_{p \in \{p \mid p = \langle v, v \rangle \land C(p)\}} \mathcal{F}(p)$ 2880 We consider two cases: 2881 Case: 2882 (11) $C(\langle v, v \rangle)$ 2883 We should show that 2884 $\mathcal{S}^{1}_{\mathsf{push}-}(v) = \mathcal{R}_{\{\langle v, v \rangle\}} \mathcal{F}(p)$ 2885 that is 2886 $S_{\text{push-}}^1(v) = \mathcal{F}(\langle v, v \rangle)$ 2887 that is straightforward from Def. 4, [4] and [11]. 2888 2889 Case: (12) $\neg C(\langle v, v \rangle)$ 2890 2891

We should show that 2892 $\mathcal{S}^{1}_{\mathsf{push}-}(v) = \mathcal{R} \mathcal{F}(p)$ 2893 2894 that is 2895 $S^1_{\text{push-}}(v) = \bot$ 2896 that is straightforward from Def. 4, [5] and [12]. 2897 2898 Inductive Case: 2899 (13) k > 12900 The induction hypothesis is: 2901 (14) $S_{\text{push}-}^{k'}(v) = \mathcal{R}_{p \in \{p \mid p \in \text{Paths}(v) \land C(p) \land \text{length}(p) < k'\}}\mathcal{F}(p)$ for all v and $k' \le k$ 2902 We should show that 2903 $\mathcal{S}_{\mathsf{push}-}^{k+1}(v) = \mathcal{R}_{p \in \{p \mid p \in \mathsf{Paths}(v) \land C(p) \land \mathsf{length}(p) < k+1\}} \mathcal{F}(p)$ 2904 We consider two cases: 2905 Case: 2906 (15) v = s2907 By Lemma 14 on [9] and [4], [5] and [10], 2908 (16) $\mathcal{S}_{\text{push}-}^{k+1}(s) = \mathcal{I}(s)$ 2909 By [4] and [9], 2910 (17) $I(s) = \mathcal{F}(\langle s, s \rangle)$ 2911 From [16] and [17], 2912 (18) $S_{\text{push}-}^{k+1}(s) = \mathcal{F}(\langle s, s \rangle)$ 2913 From [9] and [10], 2914 (19) $\mathcal{R}_{p \in \{p \mid p \in \text{Paths}(s) \land C(p) \land \text{length}(p) < k+1\}} \mathcal{F}(p) =$ 2915 $\mathcal{R}_{p \in \{p \mid p \in \text{Paths } \land \text{ tail}(p) = s \land \text{head}(p) = s \land \text{length}(p) < k+1\}} \mathcal{F}(p) =$ 2916 $\mathcal{R}_{p \in \{\langle s, s \rangle\}} \mathcal{F}(p) =$ 2917 $\mathcal{F}(\langle s, s \rangle)$ 2918 From [18] and [19], 2919 $\mathcal{S}_{\text{push-}}^{k+1}(v) = \mathcal{R}_{p \in \{p \mid p \in \text{Paths}(v) \land C(p) \land \text{length}(p) < k+1\}} \mathcal{F}(p)$ 2920 2921 Case: 2922 (20) $v \neq s$ 2923 From Def. 4, [1], [2], and [3], we have 2924 (21) $S_{\text{push-}}^{k+1}(v) = \mathcal{R}_{u \in \text{preds}(v)} \mathcal{P}\left(S_{\text{push-}}^{k}(u), \langle u, v \rangle\right)$ 2925 2926 From [21] and [14] 2927 (22) $\mathcal{S}_{\text{push-}}^{k+1}(v) = \mathcal{R}_{u \in \text{preds}(v)} \left[\mathcal{P} \left(\mathcal{R}_{p \in \{p \mid p \in \text{Paths}(u) \land C(p) \land \text{length}(p) < k\}} \mathcal{F}(p), \langle u, v \rangle \right) \right]$ 2928 In the case that the size of the set of paths is more than one, from [7], and 2929 in the case that the set of paths is singleton, from $\mathcal{R}_{\{v\}} = v$ and [8], and 2930 in the case that the set of paths is empty, from $\mathcal{R}_{\emptyset} = \bot$ and [6], 2931 we have 2932 (23) $\mathcal{P}\left(\mathcal{R}_{p \in \{p \mid p \in \text{Paths}(u) \land C(p) \land \text{length}(p) < k\}}\mathcal{F}(p), \langle u, v \rangle\right) =$ 2933 $\mathcal{R}_{p \in \{p \mid p \in \mathsf{Paths}(u) \land C(p) \land \mathsf{length}(p) < k\}} \mathcal{F}(p \cdot \langle u, v \rangle)$ 2934 After substituting [23] in [22] 2935 (24) $S_{\text{push-}}^{k+1}(v) = \mathcal{R}_{u \in \text{preds}(v)} \left[\mathcal{R}_{p \in \{p \mid p \in \text{Paths}(u) \land C(p) \land \text{length}(p) < k\}} \mathcal{F}(p \cdot \langle u, v \rangle) \right]$ 2936 that is 2937 (25) $S_{\text{push-}}^{k+1}(v) = \mathcal{R}_{p \in \{p \mid \exists u. \ p \in \text{Paths}(u) \land u \in \text{preds}(v) \land C(p) \land \text{length}(p) < k\}} \mathcal{F}(p \cdot \langle u, v \rangle)$ 2938 From [25] and Lemma 9 2939 2940

2941	(26) $S_{\text{push-}}^{k+1}(v) = \mathcal{R}_{p \in \{p \mid \exists u. \ p \in \text{Paths}(u) \land u \in \text{preds}(v) \land C(p \cdot \langle u, v \rangle) \land \text{length}(p) < k\}} \mathcal{F}(p \cdot \langle u, v \rangle)$
2942	that is
2943	(27) $\mathcal{S}_{\text{nuch}}^{k+1}(v) = \mathcal{R}_{p' \in \{p' \mid p' \in \text{Paths}(v) \land C(p') \land 0 \le \text{length}(p') \le k+1\}} \mathcal{F}(p')$
2944	From [27] and [20]
2945	$\mathcal{S}^{k+1}(v) = \mathcal{R}_{p \in \{p \mid p \in Paths(v)\} \land C(p) \land length(p) < k+1} \mathcal{F}(p),$
2946	$= push + \langle \gamma \rangle = \langle p \rangle p \in (p p \in auts(0) \land (0 p) \land (u \in gut(p) \land (1 p)) \land (1 p) \land (1 $
2947	
2948	Lemma 13.
2949	For all $\mathcal{R}, \mathcal{F}, \mathcal{C}, \mathcal{I}, \mathcal{P}, k \geq 1$ and s, v, where
2950	(1) C(p) = (head(p) = s),
2951	$(2) \forall v \in V. \ C(\langle v, v \rangle) \to I(v) = \mathcal{F}(\langle v, v \rangle)$
2952	$(3) \forall v \in V. \neg C(\langle v, v \rangle) \to I(v) = \bot$
2953	$if S_{push-}^{\kappa}(v) \neq S_{push-}^{\kappa-1}(v),$
2954	then v is reachable from s.
2955	Proof
2956	Immediate from induction on k for Def. 4.
2957	The base case is from [1], [2] and [3].
2958	
2959	
2960	Lemma 14.
2961	For all $\mathcal{R}, \mathcal{F}, C, I, \mathcal{P}, k \geq 1$ and s, v , where
2962	(1) C(p) = (head(p) = s),
2963	$(2) \forall v \in V. \ C(\langle v, v \rangle) \to I(v) = \mathcal{F}(\langle v, v \rangle)$
2964	$(3) \forall v \in V. \neg C(\langle v, v \rangle) \to I(v) = \bot$
2965	there is no cycle that contains s,
2966	then $S_{\text{push-}}^{\kappa}(s) = I(s).$
2967	Proof
2968	Immediate from induction on k for Def. 4.
2969	The inductive case is refuted by Lemma 13 and the assumption of acyclicity for s.
2970	Ţ,
2971	
2972	
2973	
2974	
2975	
2970	
2977	
2970	
2080	
2981	
2982	
2983	
2984	
2985	
2986	
2987	
2988	
2989	

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

2992	
2993	Theorem 26 (Correctness of Push (non-idempotent reduction) II).
2994	For all $\mathcal{R}, \mathcal{F}, \mathcal{C}, \mathcal{I}, \mathcal{P}$, and $k \geq 1$, if the conditions $\mathbb{C}_1 - \mathbb{C}_8$ and \mathbb{C}_{11} hold,
2995	$\mathcal{S}^{k}_{push-}(v) = \mathcal{S}pec^{k}(v)$
2996	We assume that
2997	(1) $\forall n \mathcal{P}(n+) = n$
2998	(1) $\forall n, \mathcal{N}(n, \pm) = n$ (2) $\forall n, n', \mathcal{P}(n, n') = \mathcal{P}(n', n)$
2999	(2) $\forall n, n : \mathcal{K}(n, n) = \mathcal{K}(n, n)$ (2) $\forall n, n : \mathcal{K}(n, n) = \mathcal{K}(n, n)$
3000	$(3) \forall n, n', n'' : \mathcal{K}(\mathcal{K}(n, n'), n'') = \mathcal{K}(n, \mathcal{K}(n', n''))$
3001	(4) $\forall v \in V. C(\langle v, v \rangle) \to I(v) = \mathcal{F}(\langle v, v \rangle)$
3002	$(5) \forall v \in V. \ \neg C(\langle v, v \rangle) \to I(v) = \bot$
3003	(6) $\forall e. \mathcal{P}(\bot, e) = \bot$
3004	(7) $\forall p_1, p_2 \in P, v \in V.$
3005	$tail(p_1) = tail(p_2) \rightarrow$
3006	let $u := tail(p_1)$ in
3007	$\mathcal{P}\left[\mathcal{R}(\mathcal{F}(p_1), \mathcal{F}(p_2)), \langle u, v \rangle\right] =$
3008	$\mathcal{R}\left[\mathcal{F}\left(p_{1}\cdot\langle u,v\rangle\right), \mathcal{F}\left(p_{2}\cdot\langle u,v\rangle\right)\right]$
3009	(8) $\forall p, e. \mathcal{P}(\mathcal{F}(p), e) = \mathcal{F}(p \cdot e)$
3010	(9) $\forall n, n'. \mathcal{R}(n, \mathcal{R}(\mathcal{P}(n', \langle u, v \rangle)),$
3011	$\mathcal{B}(n',\langle u,v\rangle)))=n$
3012	Form Def. 6, we have \overline{a}
3013	$Spec^{\kappa}(v) = \mathcal{R}_{p \in \{p \mid p \in \text{Paths}(v) \land C(p) \land \text{length}(p) < k\}} \mathcal{F}(p)$
3014	Dread by industion on la
3016	Proof by induction on k:
3017	k = 1
3018	K = 1 We should show that
3019	S^1 $(v) = \mathcal{R}_{r,c} (c + cc) t_{r,c} (c) + c(v) t_{r,c} (c) (c) (c)$
3020	$\sum_{push=1}^{\infty} (v) = (p \in \{p \mid p \in raths(v) \land C(p) \land length(p) < 1\}) (p)$
3021	$S^{1} \qquad (v) = \mathcal{R}_{r,c} \left(c + c + c \right) + \mathcal{L}(r) + \mathcal{L}(r)$
3022	$\bigcup_{push-(v)} - (v) \in \{p \mid p \in (v,v) \land C(p)\} $ (P) We consider two cases:
3023	Case.
3024	(10) $C(\langle n, n \rangle)$
3025	We should show that
3026	S^1 , $(v) = \mathcal{R}_{\{(v,v)\}}\mathcal{F}(p)$
3027	that is
3028	$S^{1} (v) = \mathcal{F}(\langle v, v \rangle)$
3029	that is straightforward from Def 4 [4] and [10]
3030	Case:
3031	$(11) \neg C(\langle v, v \rangle)$
3032	We should show that
3033	$\mathcal{S}^1_{\text{max}}(v) = \mathcal{R} \mathcal{F}(p)$
3034	$pusn - \chi = 0$
3035	S^1 (n) - 1
3036	$\mathcal{O}_{\text{push}-1}(\mathcal{O}) = \bot$
3037	that is straightforward from Der. 4, [5] and [11].
3038	

Inductive Case: 3040 3041 (12) k > 1The induction hypothesis is: 3042 (13) $S_{\text{push}-}^{k'}(v) = \mathcal{R}_{p \in \{p \mid p \in \text{Paths}(v) \land C(p) \land \text{length}(p) < k'\}}\mathcal{F}(p)$ for all v and $k' \le k$ 3043 3044 We should show that $\mathcal{S}_{\mathsf{push}-}^{k+1}(v) = \mathcal{R}_{p \in \{p \mid p \in \mathsf{Paths}(v) \land C(p) \land \mathsf{length}(p) < k+1\}} \mathcal{F}(p)$ 3045 3046 By Lemma 15 on [1], [2], [3], [6] and [9], we have 3047 (14) $S_{\text{push-}}^{k+1}(v) = \mathcal{R}\left[I(v), \mathcal{R}_{u \in \text{preds}(v)}\left[\mathcal{P}(S_{\text{push-}}^{k}(u), \langle u, v \rangle)\right]\right]$ 3048 From [14] and [13] 3049 (15) $\mathcal{S}_{\text{push}-}^{k+1}(v) = \mathcal{R}[$ 3050 I(v),3051 $\mathcal{R}_{u \in \text{preds}(v)} \left[\mathcal{P} \left(\mathcal{R}_{p \in \{p \mid p \in \text{Paths}(u) \land C(p) \land \text{length}(p) < k\}} \mathcal{F}(p), \langle u, v \rangle \right) \right] \right]$ 3052 In the case that the size of the set of paths is more than one, from [7], and 3053 in the case that the set of paths is singleton, from $\mathcal{R}_{\{v\}} = v$ and [8], and 3054 in the case that the set of paths is empty, from $\mathcal{R}_{\emptyset} = \bot$ and [6], 3055 we have 3056 (16) $\mathcal{P}\left(\mathcal{R}_{p \in \{p \mid p \in \text{Paths}(u) \land C(p) \land \text{length}(p) < k\}}\mathcal{F}(p), \langle u, v \rangle\right) =$ 3057 $\mathcal{R}_{p \in \{p \mid p \in \text{Paths}(u) \land C(p) \land \text{length}(p) < k\}} \mathcal{F}(p \cdot \langle u, v \rangle)$ 3058 After substituting [16] in [15] 3059 (17) $\mathcal{S}_{\text{push}-}^{k+1}(v) = \mathcal{R}[$ 3060 I(v). 3061 $\mathcal{R}_{u \in \text{preds}(v)} \left[\mathcal{R}_{p \in \{p \mid p \in \text{Paths}(u) \land C(p) \land \text{length}(p) < k\}} \mathcal{F}(p \cdot \langle u, v \rangle) \right] \right]$ 3062 that is 3063 (18) $\mathcal{S}^{k+1}_{\text{push}-}(v) = \mathcal{R}[$ 3064 3065 I(v), $\mathcal{R}_{p \in \{p \mid \exists u. \ p \in \mathsf{Paths}(u) \land u \in \mathsf{preds}(v) \land C(p) \land \mathsf{length}(p) < k\}}\mathcal{F}(p \cdot \langle u, v \rangle)]$ 3066 3067 From [18] and Lemma 9 (19) $\mathcal{S}_{\text{push-}}^{k+1}(v) = \mathcal{R}[$ 3068 3069 I(v),3070 $\mathcal{R}_{p \in \{p \mid \exists u. p \in \mathsf{Paths}(u) \land u \in \mathsf{preds}(v) \land C(p \cdot \langle u, v \rangle) \land \mathsf{length}(p) < k\}} \mathcal{F}(p \cdot \langle u, v \rangle)]$ 3071 that is 3072 (20) $\mathcal{S}_{\text{push}-}^{k+1}(v) = \mathcal{R}[$ 3073 I(v),3074 $\mathcal{R}_{p' \in \{p' \mid p' \in \text{Paths}(v) \land C(p') \land 0 < \text{length}(p') < k+1\}} \mathcal{F}(p')]$ 3075 From [4] and [5], 3076 (21) $\mathcal{R}_{p \in \{p \mid p \in \mathsf{Paths}(v) \land C(p) \land \mathsf{length}(p)=0\}}\mathcal{F}(p) = \mathcal{I}(v)$ 3077 From [20] and [10], 3078 (22) $\mathcal{S}_{\text{push+}}^{k+1}(v) = \mathcal{R}[$ 3079 $\mathcal{R}_{p \in \{p \mid p \in \text{Paths}(v) \land C(p) \land \text{length}(p)=0\}}\mathcal{F}(p),$ 3080 $\mathcal{R}_{p \in \{p \mid p \in \text{Paths}(v) \land C(p) \land 0 < \text{length}(p) < k+1\}} \mathcal{F}(p)$] 3081 that is 3082 $\mathcal{S}_{\text{push+}}^{k+1}(v) = \mathcal{R}_{p \in \{p \mid p \in \text{Paths}(v) \land C(p) \land \text{length}(p) < k+1\}} \mathcal{F}(p),$ 3083 3084 3085 3086 3087

Anon.

LEMMA 15. 3088 For all $\mathcal{R}, \mathcal{F}, C, I, \mathcal{P}, k \geq 1$ if the conditions $\mathbb{C}_1 - \mathbb{C}_9$ hold, 3089 $\mathcal{S}_{\mathsf{push}-}^{k}(v) = \mathcal{R}\left[I(v), \ \mathcal{R}_{u \in \mathsf{preds}(v)}\left[\mathcal{P}(\mathcal{S}_{\mathsf{push}-}^{k-1}(u), \langle u, v \rangle)\right]\right]$ 3090 3091 3092 Proof. 3093 We assume that 3094 (1) $\forall n. \mathcal{R}(n, \perp) = n$ 3095 (2) $\forall n, n'. \mathcal{R}(n, n') = \mathcal{R}(n', n)$ 3096 (3) $\forall n, n', n''$. $\mathcal{R}(\mathcal{R}(n, n'), n'') = \mathcal{R}(n, \mathcal{R}(n', n''))$ 3097 (4) $\forall e. \mathcal{P}(\bot, e) = \bot$ 3098 (5) $\forall n, n'. \mathcal{R}(n, \mathcal{R}(\mathcal{P}(n', \langle u, v \rangle)),$ 3099 $\mathcal{B}\left(n',\langle u,v\rangle\right)) = n$ 3100 3101 Proof by induction on *k*: 3102 Base Case: 3103 (6) k = 13104 By Def. 4, 3105 (7) $\forall u. \mathcal{S}^0_{\text{push}-}(u) = \bot$ 3106 (8) $\forall u. S^{1}_{\text{push-}}(u) = I(u)$ From [7], [4] and [1], 3107 3108 (9) $\mathcal{R}\left[I(v), \mathcal{R}_{u \in \text{preds}(v)}\left[\mathcal{P}(\mathcal{S}_{\text{push-}}^{0}(u), \langle u, v \rangle)\right]\right] = I(v)$ 3109 3110 From [9] and [8], $S_{\mathsf{push-}}^{1}(u) = \mathcal{R}\left[I(v), \ \mathcal{R}_{u \in \mathsf{preds}(v)}\left[\mathcal{P}(S_{\mathsf{push-}}^{0}(u), \langle u, v \rangle)\right]\right]$ 3111 3112 3113 Inductive Case: 3114 The induction hypothesis is 3115 (10) $S_{\text{push-}}^{k}(v) = \mathcal{R}\left[I(v), \mathcal{R}_{u \in \text{preds}(v)}\left[\mathcal{P}(S_{\text{push-}}^{k-1}(u), \langle u, v \rangle)\right]\right]$ for all $k' \leq k$ 3116 We show that 3117 $\mathcal{S}_{\mathsf{push-}}^{k+1}(v) = \mathcal{R}(\mathcal{I}(v), \mathcal{R}_{u \in \mathsf{preds}(v)} \left[\mathcal{P}(\mathcal{S}_{\mathsf{push-}}^k(u), \langle u, v \rangle) \right])$ 3118 3119 From Def. 4, we have that (11) $\mathcal{S}_{\text{push}-}^{k+1}(v) \coloneqq S_n$ 3120 3121 (12) $\{u_1, ..., u_n\} = u \in \left\{u \mid u \in \operatorname{preds}(v) \land \mathcal{S}_{\operatorname{push}}^k(u) \neq \mathcal{S}_{\operatorname{push}}^{k-1}(u)\right\}$ 3122 (13) $S_0 \coloneqq \mathcal{S}_{push-}^k(v)$ 3123 3124 (14) $S_{i+1} \coloneqq \mathcal{R}(\mathcal{R}(S_i,$ 3125 $\mathcal{B}\left(\mathcal{S}_{\mathsf{push}-}^{k-1}(u_i), \langle u_i, v \rangle\right)$ 3126 $\mathcal{P}\left(\mathcal{S}_{\mathsf{push-}}^{k}(u_{i}),\langle u_{i},v\rangle\right)$ 3127 3128 From [11]-[14], and [2] and [3], we have (15) $\mathcal{S}_{\text{push+}}^{k+1}(v) = \mathcal{R}(\mathcal{S}_{\text{push-}}^{k}(v)),$ 3129 3130 $\mathcal{R}_{u \in \{u \mid u \in \text{preds}(v) \land \mathcal{S}_{\text{push-}}^{k}(u) \neq \mathcal{S}_{\text{push-}}^{k-1}(u)\}} \mathcal{R}(\mathcal{B}\left[\mathcal{S}_{\text{push-}}^{k-1}(u_{i}), \langle u_{i}, v \rangle\right] \mathcal{P}\left[\mathcal{S}_{\text{push-}}^{k}(u_{i}), \langle u_{i}, v \rangle\right])$ 3131 3132 3133 3134 From [15] and [10], we have 3135 3136

(16)
$$\mathcal{S}_{\text{push+}}^{k+1}(v) = \mathcal{R}(\mathcal{R}\left[I(v), \mathcal{R}_{u \in \text{preds}(v)}\left[\mathcal{P}(\mathcal{S}_{\text{push-}}^{k-1}(u), \langle u, v \rangle)\right]\right],$$

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

- $\begin{aligned} \mathcal{R}_{u \in \{u \mid u \in \operatorname{preds}(v) \land \mathcal{S}_{\operatorname{push-}}^{k}(u) \neq \mathcal{S}_{\operatorname{push-}}^{k-1}(u)\}} \mathcal{R}(\\ \mathcal{B}\left[\mathcal{S}_{\operatorname{push-}}^{k-1}(u_{i}), \langle u_{i}, v \rangle\right] \\ \mathcal{P}\left[\mathcal{S}_{\operatorname{push-}}^{k}(u_{i}), \langle u_{i}, v \rangle\right] \end{aligned}$

$$\mathcal{P}\left[\mathcal{S}_{\mathsf{push}-}^{k}(u_{i}),\langle u_{i},v\rangle\right]$$

that is

$$\begin{array}{ll} & (17) \ \ \mathcal{S}_{push+}^{k+1}(v) = \mathcal{R}(\mathcal{R}(I(v),\mathcal{R}(v))) \\ & (17) \ \ \mathcal{S}_{push+}^{k+1}(v) = \mathcal{R}(\mathcal{R}(v),\mathcal{R}(v)) \\ & (17) \ \ \mathcal{S}_{push+}^{k+1}(v) = \mathcal{R}(\mathcal{R}(v),\mathcal{R}(v)) \\ & (18) \ \ \mathcal{S}_{push+}^{k+1}(v) = \mathcal{R}(\mathcal{I}(v),\mathcal{R}(v)) \\ \end{array}$$

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

$$\mathcal{R}_{u \in \{u \mid u \in \operatorname{preds}(v) \land \mathcal{S}_{\operatorname{push}^{k}}^{k}(u) \neq \mathcal{S}_{\operatorname{push}^{k-1}}^{k-1}(u)\}} \mathcal{P} \left[\mathcal{S}_{\operatorname{push}^{k}}^{k}(u_{i}), \langle u_{i}, v \rangle \right]) \right]$$

that is
(19)
$$S_{\text{push+}}^{k+1}(v) = \mathcal{R}(I(v), \mathcal{R}(\mathcal{R}_{u \in \{u \mid u \in \text{preds}(v) \land S_{\text{push-}}^{k}(u) = S_{\text{push-}}^{k-1}(u)\}} \left[\mathcal{P}(S_{\text{push-}}^{k}(u), \langle u, v \rangle)\right],$$

 $\mathcal{R}_{u \in \{u \mid u \in \text{preds}(v) \land S_{\text{push-}}^{k}(u) \neq S_{\text{push-}}^{k-1}(u)\}} \mathcal{P}\left[S_{\text{push-}}^{k}(u_{i}), \langle u_{i}, v \rangle\right])$

that is (20) §

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

b)

3186 4.4.5 Termination 3187

THEOREM 27 (TERMINATION). 3188 For all $\mathcal{R}, \mathcal{F}, and \mathcal{C},$ 3189 if the graph is acyclic or the condition \mathbb{C}_{10} holds, then there exists k such that for every $k' \geq k$ 3190 $Spec^{k'}(v) = Spec(v).$ 3191 3192 Proof. 3193 We assume that 3194 (1) $Spec(v) = \mathcal{R}_{p \in \{p \mid p \in Paths(v) \land C(p)\}} \mathcal{F}(p)$ 3195 (2) $Spec^{k}(v) = \mathcal{R}_{p \in \{p \mid p \in \text{Paths}(v) \land C(p) \land \text{length}(p) < k\}} \mathcal{F}(p)$ 3196 (3) The graph is acyclic or 3197 \mathbb{C}_{10} : $\mathcal{R}(\mathcal{F}(p), \mathcal{F}(\operatorname{simple}(p))) = \mathcal{F}(\operatorname{simple}(p))$ 3198 Let 3199 (4) *l* be the longest simple path to *v* (that satisfies *C*). 3200 Let 3201 (5) $P^{l+1} = \{p \mid p \in \text{Paths}(v) \land C(p) \land \text{length}(p) < l+1\}$ 3202 (6) $P^{l+i} = \{p \mid p \in \text{Paths}(v) \land C(p) \land \text{length}(p) < l+i\}, i > 1$ 3203 (7) $P = \{p \mid p \in \text{Paths}(v) \land C(p)\}$ 3204 From [2], [5] and [6], we have 3205 (8) $Spec(v) = \mathcal{R}_P \mathcal{F}(p)$ 3206 (9) $Spec^{l+1}(v) = \mathcal{R}_{pl+1} \mathcal{F}(p)$ 3207 (10) $Spec^{l+i}(v) = \mathcal{R}_{pl+i} \mathcal{F}(p)$ 3208 From [4], [7], and [5], 3209 (11) No path in $P \setminus P^{l+1}$ is simple. 3210 (12) No path in $P \setminus P^{l+i}$ is simple. 3211 3212 From [3], we consider two cases: 3213 Case: 3214 (13) The graph is acyclic. 3215 From [11], [12] and [13], we have 3216 (14) $P^{l+1} = P^{l+i} = P$ 3217 Thus, from [8], [9] and [10], for k' = l + 1, for all $k' \ge k$, we have 3218 $Spec^{k'}(v) = Spec(v)$ 3219 3220 Case: 3221 (15) $\forall p. \mathcal{R}(\mathcal{F}(p), \mathcal{F}(\operatorname{simple}(p))) = \mathcal{F}(\operatorname{simple}(p))$ 3222 From [11] and [4], we have 3223 (16) $\forall p. p \in P \setminus P^{l+1} \rightarrow \text{length}(\text{simple}(p_1)) < l+1$ 3224 From [7], we have 3225 (17) $\forall p. p \in P \setminus P^{l+1} \to p \in \text{Paths}(v)$ 3226 (18) $\forall p. p \in P \setminus P^{l+1} \to C(p)$ 3227 From [17], we have 3228 (19) $\forall p. p \in P \setminus P^{l+1} \rightarrow \text{simple}(p) \in \text{Paths}(v)$ 3229 By Lemma 16 and [18], 3230 (20) $\forall p. p \in P \setminus P^{l+1} \rightarrow C(\operatorname{simple}(p))$ 3231 From [19], [20], [16] and [5] 3232 (21) $\forall p. p \in P \setminus P^{l+1} \rightarrow \text{simple}(p) \in P^{l+1}$ 3233 3234

From [15], (22) $\forall p. p \in P \setminus P^{l+1} \to \mathcal{R}(\mathcal{F}(p), \mathcal{F}(\operatorname{simple}(p))) = \mathcal{F}(\operatorname{simple}(p))$ From [21] and [22], (23) $\forall p. p \in P \setminus P^{l+1} \to \mathcal{R}(\mathcal{F}(p), \mathcal{R}_{pl+1} \mathcal{F}(p)) = \mathcal{R}_{pl+1} \mathcal{F}(p)$ therefore (24) $\mathcal{R}(\mathcal{R}_{p \setminus pl+1} \mathcal{F}(p), \mathcal{R}_{pl+1} \mathcal{F}(p)) = \mathcal{R}_{pl+1} \mathcal{F}(p)$ that is (25) $\mathcal{R}_P \mathcal{F}(p) = \mathcal{R}_{pl+1} \mathcal{F}(p)$ From [25], [8] and [9], we have (26) $Spec(v) = Spec^{l+1}(v)$ Similarly, for every k > l + 1, we can prove that (27) $Spec^k(v) = Spec^{l+1}(v)$ From [26] and [27], we have that for $k' \ge l + 1$, $Spec^{k'}(v) = Spec(v)$ LEMMA 16. $\forall p. C(p) \leftrightarrow C(simple(p))$ 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) = TrueStraightforward by True \leftrightarrow True

3284 5 Implementation

³²⁸⁵ 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.

```
3333
      class Engine<graph, gather_reducer,</pre>
3334
         message_reducer> {
                                                            int main() {
3335
       void run() {
                                                             Graph<vertex_type, edge_type> g;
         active: Set of signaled vertices
3336
                                                             g.load();
         next_active = \emptyset;
3337
                                                             g.transform_vertices(initialize);
         while(active != Ø) {
3338
                                                             g.transform_edges(init_edge);
          par_for(v ∈ signalled) {
3339
           init(v, msg);
3340
                                                             Engine engine = new Engine<g,
           dir_type gd = gather_edges(v);
                                                                  gather_reducer,
3341
           par_for(e \in edges(v, gd))
                                                                  message_reducer>
3342
            gv = gather(v, e);
3343
           apply(v, gv);
                                                             engine.map_reduce_edges(signal_vertices);
3344
           dir_type sd = scatter_edges(v);
                                                             // engine.signal_all();
3345
           par_for(e \in edges(v, sd))
                                                             engine.run();
            scatter(v, e);
3346
                                                             // T aggregated_value =
          } }
3347
                                                                  engine.map_reduce_vertices<T>(transform);
                                                             //
          active = next_active;
3348
                                                            }
          next_active = Ø;
3349
       } };
3350
```

Fig. 22. PowerGraph Runtime

5.2 PowerGraph 3355

PowerGraph is a distributed graph processing system that provides a shared-memory programming 3356 abstraction. It efficiently processes power-law graphs by incorporating a vertex-cut strategy for 3357 balanced workload distribution, and by parallelizing vertex computations across edges. It achieves 3358 this by splitting vertex-computations across three steps: gather, apply, and scatter. Figure 22 3359 shows PowerGraph's iterative processing model. The run() method processes a set of vertices in 3360 each iteration by invoking five functions (marked in blue). The gather() function iterates through 3361 edges of a vertex (incoming, outgoing, both or none, as defined by gather_edges()) to aggregate 3362 the values from its neighbors. The apply() function computes a new value of the vertex based on 3363 the aggregated value from the gather step. Finally, the scatter() function iterates through edges 3364 of a vertex (incoming, outgoing, both or none, as defined by scatter_edges()) to propagate its 3365 new value to its neighbors. 3366

In each iteration, the set of vertices to be processed are identified via explicit vertex-signaling 3367 mechanism. Typically, if a vertex's value changes, it 'signals' its neighbors in the scatter() 3368 function so that they get processed in the subsequent iteration. For the first iteration, the set of 3369 vertices to be processed are signalled before invoking the run() method (as shown in main()). 3370

Apart from iterative processing, PowerGraph also provides capabilities for transforming and 3371 reducing vertex (and edge) values. The map_reduce_vertices() function shown in main() can 3372 be used to perform vertex-based reductions. 3373

3374 3375

3376

3351

3352 3353 3354

Mapping Synthesized Functions.

PowerGraph allows expressing graph computations in pull mode (Figure 23) and in push mode 3377 (Figure 24). In pull mode, the propagation of values across edges occurs in the gather step, and the 3378 values propagated to a vertex (or 'pulled by a vertex') in this step are passed through an aggregator 3379 as defined in struct reducer. In push mode, value propagation occurs in the scatter step and 3380

67

```
3382
      struct reducer {
3383
       VValueType value;
                                                          void apply(vertex_type& v, reducer& red_gv) {
3384
       reducer& operator+=(reducer& other) {
                                                           changed = false;
        value = reduce(value, other.value);
                                                           if(reduce(red_gv.value, v.data()) !=
3385
        return *this;
                                                                      v.data()) {
3386
       } }
                                                            v.data() = red_gv.value;
3387
                                                            changed = true;
3388
      bool changed = false;
                                                           }
3389
      void init(vertex_type& v, empty_type& m) { }
                                                          }
3390
3391
      dir_type gather_edges(vertex_type& v) {
                                                          dir_type scatter_edges(vertex_type& v) {
3392
       return in_edges; }
                                                           return changed ? out_edges : no_edges;
3393
                                                          }
3394
      reducer gather(vertex_type& v, edge_type& e) {
3395
       if (e.source().data() != none) {
                                                          void scatter(vertex_type& v,
        return propagate(e.source(), e);
3396
                                                                         edge_type& e) {
       } else {
                                                           signal(e.target());
3397
        return none;
                                                          }
3398
       } }
3399
3400
                                              Fig. 23. PowerGraph Pull
3401
3402
3403
      struct reducer {
                                                          void apply(vertex_type& v,
3404
       VValueType value;
                                                                      reducer& red_gv) {
       reducer& operator+=(reducer& other) {
                                                           changed = false;
3405
        value = reduce(value, other.value);
                                                           if(reduce(msg.value, v.data()) !=
3406
        return *this;
                                                                      v.data()) {
3407
                                                            v.data() = msg.value;
       } }
3408
                                                            changed = true;
3409
      bool changed = false;
                                                           } }
3410
      reducer msg;
3411
      void init(vertex_type& v,
                                                          dir_type scatter_edges(vertex_type& v) {
3412
                                                           return changed ? out_edges : no_edges;
                 empty_type& m) {
3413
                                                          }
       msg = m;
3414
      }
3415
                                                          void scatter(vertex_type& v,
      dir_type gather_edges(vertex_type& v) {
                                                                         edge_type& e) {
3416
                                                           VValueType new_val = propagate(v, e);
       return no_edges;
3417
                                                           if(reduce(new_val, e.target().data()) !=
      }
3418
                                                                      e.target().data()) {
3419
      reducer gather(vertex_type& v,
                                                            signal(e.target(), new_val);
3420
                      edge_type& e) { }
                                                           } }
3421
3422
                                              Fig. 24. PowerGraph Push
3423
3424
      the values propagated to a vertex (or 'pushed to a vertex') in this step are passed through the
3425
```

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 directly directl

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.

³⁴⁸⁰ 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.

³⁴⁸⁷ 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,
                                                             while(!frontier.isEmpty()) {
3503
           vertexSubset U, func f, func c) {
                                                             next_frontier = edgeMap(g,
3504
       vertexSubset out = \emptyset;
                                                                 frontier, edgeFunction,
3505
       par_for(v \in U)
                                                                condFunction);
3506
        par_for(ngh ∈ out_neighbors(v))
                                                             frontier.del();
3507
         if(c(ngh) && f(v, ngh, w(ngh)))
                                                             frontier = next_frontier;
          out = out.insert(ngh);
3508
                                                             }
       return out;
                                                             frontier.del();
3509
                                                            }
      }
3510
3511
                                                            bool edgeFunction(VIdType s, VIdType d,
      void compute(graph g) {
3512
       VValueType* values =
                                                                                EWeightType w) {
3513
              new VValueType[g.n];
                                                             return reduce(&values[d],
3514
       par_for(VIdType i=0;i<n;i++)</pre>
                                                                    propagate(s, EdgeType(s, d, w)));
3515
        values[i] = initialize(i);
                                                            }
3516
       vertexSubset frontier(n,src);
3517
       // vertexSubset frontier(n, n,
                                                            bool condFunction(VIdType d)
3518
       11
                        [1, 1, .., 1]);
                                                             { return true; }
3519
3520
                                             Fig. 25. Ligra Runtime & Push
3521
```

3500

3501

3502
3529 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.

³⁵³⁶ 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.

```
int main() {
3549
      template<typename EDGE_MAP>
                                                             WGraph g;
3550
      vertexSubset edgeset_apply(WGraph g,
                                                             g.load();
            vertexSubset U, EDGE_MAP f) {
3551
                                                             VValueType* values =
       vertexSubset out = \emptyset;
3552
                                                                    new VValueType[g.n];
       par_for(v \in U)
3553
                                                             par_for(VIdType i=0;i<n;i++)</pre>
        par_for(ngh ∈ out_neighbors(v))
3554
                                                              values[i] = initialize(i);
          if(f(v, ngh, w(ngh)))
3555
                                                             vertexSubset frontier(n,src);
          out = out.insert(ngh);
3556
                                                             //vertexSubset frontier(n,n);
       return out;
                                                             addVertex(frontier, src) ;
3557
      }
                                                             while(!frontier.isEmpty()) {
3558
                                                              next_frontier =
3559
      struct edgeMap {
                                                                edgeset_apply(edges, frontier, edgeMap());
3560
       bool operator(NodeID s, NodeID d, int w) {
                                                              frontier.del();
3561
         return reduce(&values[d],
                                                              frontier = next_frontier;
               propagate(s, EdgeType(s, d, w)));
3562
                                                             }
       }
3563
                                                             frontier.del();
      }
3564
                                                            }
3565
3566
```

Fig. 26. Graphit Runtime & Push

3578 5.5 Gemini

3579 Gemini is a NUMA-aware, distributed, high-performance graph processing system. It extracts 3580 parallelism across multicores by partitioning threads across NUMA nodes, and uses MPI for 3581 coordination across machines. It incorporates a hybrid push-pull processing model that dynamically 3582 switches between pull mode and push mode depending on the number of active vertices. The pull 3583 mode is performed when number of active vertices is large (based on a threshold), and it effectively 3584 iterates over all the incoming edges of a vertex to compute its next value. On the other hand, the 3585 push mode is performed when number of active vertices is small and it iterates over all the outgoing 3586 edges of a vertex to compute their next value.

3587 Similar to Ligra, we show process_edges() in Figure 27 since our path-based computations 3588 operate on edges only. As we can see, process_edges() accepts four user-defined callbacks along 3589 with a bitmask indicating the set of active vertices. The bitmask is first checked to determine sparsity 3590 of the iteration, based on which, either the first two callbacks are invoked (if sparse), or the other two 3591 call backs are invoked (if dense). The sparse_signal and dense_signal callbacks determine the 3592 value to be propagated from/to a vertex to/from its outgoing and incoming neighbors respectively. 3593 These values are maintained in form of messages, that are shuffled and sorted across NUMA nodes 3594 and machines. Then, the sparse_slot and dense_slot callbacks compute the new vertex value 3595 based on the propagated values (or messages) from sparse_signal and dense_signal respectively, 3596 and also activate neighboring vertices to be processed in the next iteration. It is interesting to note 3597 that iterating over the incoming and outgoing edges is performed by the user-defined callbacks, as 3598 opposed to the runtime as achieved in PowerGraph and Ligra. 3599

³⁶⁰⁰ 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.

3606

3607

3608

3609

```
3627
3628
      VertexId process_edges(func sparse_signal,
                                                            while(num_active_vertices > 0) {
3629
        func sparse_slot, func dense_signal,
                                                             active_out->clear();
3630
        func dense_slot, Bitmap* active) {
                                                             num_active_vertices = g->process_edges(
       sparse = compute_sparsity(active);
                                                              [&](VertexId src){
3631
       if(sparse) {
                                                               g->emit(src, values[src]);
3632
        par_for(VertexId v ∈ active)
                                                              },
3633
                                                              [&](VertexId src, VValueType msg, AdjList out_nbrs) {
         sparse_signal(v);
3634
        exchange_messages();
                                                               VertexId activated = 0:
3635
        par_for(msg ∈ messages) {
                                                               for (AdjUnit* ptr ∈ out_nbrs) {
3636
         VertexId source = message.vertex;
                                                                VertexId dst = ptr->neighbour;
3637
         sparse_slot(source, message.msg_data, outAdjList[v]);if(reduce(&values[dst], propagate(msg,
3638
        }
                                                                          EdgeType(src, dst, ptr->edge_data)))) {
3639
       } else {
                                                                 active_out->set_bit(dst);
3640
        activated += 1;
3641
         dense_signal(v, inAdjList[v]);
                                                                }
3642
        exchange_messages();
                                                               }
        par_for(msg ∈ messages) {
                                                               return activated;
3643
         VertexId target = message.vertex;
                                                              },
3644
         dense_slot(target, message.msg_data);
                                                              [&](VertexId dst, AdjList in_nbrs) {
3645
        }
                                                               VValueType msg = none;
3646
                                                               for (AdjUnit* ptr ∈ in_nbrs) {
       }
3647
      }
                                                                VertexId src = ptr->neighbour;
3648
                                                                reduce(&msg, propagate(values[src],
3649
                                                                       EdgeType(src, dst, ptr->edge_data)));
      int main() {
3650
       Graph g;
                                                               }
3651
       g.load();
                                                               if (msg != none) g->emit(dst, msg);
3652
       values = g->alloc_vertex_array<VValueType>();
                                                              },
3653
       VertexSubset* active_in = g->alloc_vertex_subset(); [&](VertexId dst, VValueType msg) {
       VertexSubset* active_out = g->alloc_vertex_subset(); if(reduce(&values[dst], msg)) {
3654
                                                                active_out->set_bit(dst);
3655
       for(VertexId i=0; i<g->vertices; ++i)
                                                                return 1:
3656
        values[i] = initialize(i);
                                                               }
3657
                                                               return 0;
3658
       active_in->clear();
                                                              },
3659
       active_in->set_bit(src);
                                                              active_in
3660
       VertexId num_active_vertices = 1;
                                                             ):
3661
       // active_in->fill();
                                                             swap(active_in, active_out);
3662
       // VertexId num_active_vertices = graph->vertices; }
3663
                                                           }
3664
3665
3666
                                     Fig. 27. Gemini Hybrid Push-Pull Runtime
3667
3668
3669
```

3676 5.6 GridGraph

3677 GridGraph is an out-of-core disk-based graph processing system. It maintains the graph in a 3678 2D grid layout that resides on disk, and uses a streaming partition based processing model to 3679 sequentially accesses disk partitions. Figure 28 shows stream_edges and stream_vertices that 3680 are used to process the graph. The stream_edges function processes active set of edges by reading 3681 the corresponding partitions from disk one-by-one and invoking the user-defined process function 3682 on the edge. The stream_vertices function invokes a user-defined function on active vertices 3683 (similar to MAP operation). It is interesting to note that both these methods take care of disk 3684 operations so that the user-defined functions can focus solely on edge and vertex computations. 3685

3689 Mapping Synthesized Functions.

Similar to Ligra, we express our path computations on GridGraph in push mode. The main func tion 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).

```
return 0;
3696
      void stream_edges(func process, Bitmap* active) {
                                                              });
3697
       for(partition p \in partitions) \{
3698
        if(p ∉ active)
                                                              active_out->clear();
3699
         continue:
                                                              active_out->set_bit(src);
3700
         for(Edge e \in p)
                                                              VertexId num_active_vertices = 1;
3701
         if(e.source ∈ active)
                                                              // active_out->fill();
3702
          process(e);
                                                              // VertexId num_active_vertices = g.vertices;
       }
3703
      }
                                                              while (num_active_vertices > 0) {
3704
                                                               swap(active_in, active_out);
3705
      void stream_vertices(func process, Bitmap* active) { active_out->clear();
3706
       par_for(VertexId v \in V) {
                                                               active_vertices = g.stream_edges<VertexId>([&]
3707
        if(v \in active)
                                                                          (Edge& e) {
3708
                                                                if (reduce(&vertex_values[e.target],
         process(v);
3709
                                                                   propagate(
       }
3710
      }
                                                                      e.source,
3711
                                                                      EdgeType(e.source, e.target, e.w))
3712
       int main() {
                                                                   )) {
3713
       Graph g(load_path);
                                                                 active_out->set_bit(e.target);
       Bitmap* active_in = g.alloc_bitmap();
                                                                 return 1;
3714
       Bitmap* active_out = g.alloc_bitmap();
                                                                }
3715
                                                                return 0:
       vertex_values.init(vertex_path, g.vertices);
3716
                                                               }, active_in);
3717
       g.stream_vertices<VertexId>([&](VertexId i) {
                                                              } }
3718
        vertex_values[i] = initialize(i);
3719
3720
3721
                                              Fig. 28. GridGraph Runtime
3722
3723
```

74

3686 3687 3688

3694 3695

3725 5.7 Path-based Reduction Synthesis

```
3726
       //NWR usecase
3727
       struct VValueType{
3728
        uint32_t first;
3729
       uint32_t second;
3730
       };
3731
       VValueType reduce(const VValueType a,
3732
         const VValueType b) {
3733
        bool r = 0;
3734
        VValueType c;
3735
        VValueType w;
3736
        do {
        c = a;
3737
         w = c;
3738
         if (b.first < c.first) {</pre>
3739
         w.first = b.first;
3740
         } else {
3741
         if (b.first > c.first) {
3742
          w.first = c.first;
3743
          }
3744
3745
         }if (b.second > c.second) {
         w.second = b.second;
3746
3747
         } else {
3748
         if (b.second < c.second) {</pre>
3749
          w.second = c.second;
3750
          }
3751
         }
3752
        } while(((b.second > c.second ||
3753
         b.first < c.first) &&</pre>
3754
            !(r=cas(a,c,w))));
3755
        return r;
3756
       }
```

3757 3758

3759

//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) {</pre> w.first = b.first; } else { if (b.first > c.first) { w.first = c.first; } }if (b.second < c.second) {</pre> w.second = b.second; } else { if (b.second > c.second) { w.second = c.second; } } } while(((b.second < c.second ||</pre> b.first < c.first) &&</pre> !(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.

Anon.

```
3774
      //BFS usecase
3775
      struct VValueType{
                                                            //SP usecase
3776
       uint32_t first;
                                                            struct VValueType{
       uint32_t second;
3777
                                                            uint32_t first;
      }:
3778
                                                            };
      VValueType reduce(const VValueType a,
3779
                                                            VValueType reduce(const VValueType a,
        const VValueType b) {
3780
                                                              const VValueType b) {
       bool r = 0;
3781
                                                             bool r = 0;
       VValueType c;
3782
                                                             VValueType c;
       VValueType w;
                                                             VValueType w;
3783
       do {
                                                             do {
3784
        c = a;
                                                              c = a;
3785
        w = c;
                                                              w = c;
3786
        if (b.first < c.first) {</pre>
                                                              if (b.first < c.first) {</pre>
3787
        w.first = b.first;
                                                              w.first = b.first;
         w.second = b.second;
3788
                                                              } else {
        } else {
3789
                                                               if (b.first > c.first) {
         if (b.first > c.first) {
3790
                                                               w.first = c.first;
          w.first = c.first;
3791
                                                               }
          w.second = c.second;
3792
                                                              }
         }
3793
                                                             } while((b.first < c.first &&</pre>
        }
                                                              !(r=cas(a,c,w))); return r;
3794
       } while((b.first < c.first &&</pre>
                                                            }
3795
        !(r=cas(a,c,w))); return r;
3796
      }
                                                            //WP usecase
3797
                                                            struct VValueType{
3798
      //CC usecase
                                                             uint32_t first;
      struct VValueType{
3799
                                                            };
       uint32_t first;
3800
                                                            VValueType reduce(const VValueType a,
      };
3801
                                                              const VValueType b) {
      VValueType reduce(const VValueType a,
3802
                                                             bool r = 0;
        const VValueType b) {
3803
                                                             VValueType c;
       bool r = 0;
                                                             VValueType w;
3804
       VValueType c;
                                                             do {
3805
       VValueType w;
                                                              c = a;
3806
       do {
                                                              w = c;
3807
        c = a;
                                                              if (b.first > c.first) {
3808
        w = c;
                                                              w.first = b.first;
        if (b.first > c.first) {
3809
                                                              } else {
         w.first = b.first;
3810
                                                              if (b.first < c.first) {</pre>
        } else {
3811
                                                               w.first = c.first;
         if (b.first < c.first) {</pre>
3812
                                                               }
          w.first = c.first;
3813
                                                              }
         }
3814
                                                             } while((b.first > c.first &&
        }
                                                              !(r=cas(a,c,w))); return r;
3815
       } while((b.first > c.first &&
                                                            }
3816
        !(r=cas(a,c,w))); return r;
3817
      }
3818
3819
                           Fig. 30. Generated atomic reduce functions for simple use-cases.
3820
3821
3822
```

3823	//WSP usecase
3824	<pre>struct VValueType{</pre>
3825	uint32_t first;
3826	uint32_t second;
3827	};
3828	
3829	<pre>VValueType reduce(const VValueType a,</pre>
3830	<pre>const VValueType b) {</pre>
3831	bool r = 0;
3832	VValueType c;
2022	VValueType w;
3834	do {
2025	c = a;
3835	w = c;
3836	if (b.first < c.first) {
3837	w.first = b.first;
3838	w.second = b.second;
3839	f else (
3840	If $(D.111St > C.111St)$ {
3841	w. $rirst = c. rirst,$
3842	w.secolu = c.secolu,
3843	J
3844	<pre>}if (c.first == b.first) {</pre>
3845	w.first = c.first;
3846	<pre>w.second = std::max(b.second, c.second);</pre>
3847	}
3848	
3849	<pre>} while(((b.first < c.first </pre>
3850	(c.first == b.first &&
3851	b.second > c.second)) &&
3852	!(r=cas(a,c,w))));
3853	return r;
3854	}
3855	
3856	Fig. 31. Generated atomic reduce function for WSP
3857	usecase. The rule FPNEST is used to generate atomic
3858	reduce functions for WSP usecase.
3859	
3860	
3861	
3862	
3863	
3864	
3865	
3866	
3867	
2020	
2840	
2870	
3070	
30/1	

3872 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.
 3877
 - 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

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.

3970 6.2 The Effect of Fusion

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

3973	Prog	Input		Ligra		G	ridGrapl	1		Gemini		Powe	erGraph	(Push)	Powe	rGraph ((Pull)	Gra	phIt (Pu	ısh)
3974	1105.	mput	S	H	R	S	H	R	S	H	R	S	Η	R	S	H	R	S	H	R
3975		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
3976	DRR	TW	-	-	-	82	215	2.6	7	16	2.2	61	184	3	107	392	3.6	12	41	3.3
3770		TM	-	-	-	130	325	2.5	33	110	3.3	94	313	3.3	223	760	3.4	202	345	1.7
3977		FK	-	-	-	225	404	2	2/	00	2.5	202	520	2.5	297	1095	5.0	-	-	-
3978		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
2070	Trust	TW	-	-	-	2413	2433	1	11.9	16	1.3	-	-	-	151	392	2.6	23	48	2.1
3979		FR	-	-	-	540	620	1.0	24 7965	11105	1.4	364	419	1.1	367	1003	27	940	570	- 0.4
3980						010	020		1						1 507	1 1005			<u> </u>	
3081		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
5701	LTrust	TM	-	-	-	142	186	1.9	10	15.5	1.5	281	324	1.2	-	-	-	324	679	1.0
3982		FR	-	-	-	584	1048	1.8	5300	7315	1.3	389	442	1.2	-	-	-	-	-	-
3983																			<u> </u>	
2094																				
3984		Ligı	a Gr	id Ge1	nini I	PG P	G Grap	hIt	Ligra	GridGe	mini P	G P	G Gra	phIt	Ligra	Grid Ge	mini	PG	PG Gr	aphIt
3985	1(<u> </u>	Gra	ph	P	ush Pı	ıll			Graph	Pu	ish P	ull		C	Fraph	I	Push	Pull	



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 $2.1 \times$ speedup across different frameworks and input graphs.

4019 6.3 Fusion Types

20																		
21	Lico coco	aca Innut		Ligra			GridGraph			Gemini			PowerGraph (Push)			PowerGraph (Pull)		
	Use-case	mput	Н	S	R	Н	S	R	Н	S	R	Н	S	R	Н	S	R	
22		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	
23	WSP	TW				37.4	19.6	1.9	10	6.5	1.5	47.2	27	1.74	130.9	69.6	1.9	
24		TM				71	37.4	1.9	14.3	9.3	1.5	78.7	45.3	1.7	199.2	92.5	2.1	
25		FR				142.6	81	1.7	15.7	9.9	1.5	116.1	59.6	1.9	237.5	116.8	2	
23		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	
26	Radius	TW				40.8	21	1.9	38.6	24.6	1.6	55	45.7	1.2	156	80.6	1.9	
27		TM				70.2	35.6	1.9	66.6	39.6	1.6	84.3	67.7	1.2	237	130.6	1.8	
28		FR				151.4	89.4	1.7	218.2	104.2	2	115	75.9	1.5	234	126	1.8	
29		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	
	NWR	TW				37.8	20.8	1.8	14.4	6.7	2.1	52.7	23.4	2.2	132.7	63	2.1	
30		TM				62	41	1.5	22	11	2	76	38.1	2	200	97.1	2	
31		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×.



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.

4117 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.



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.

# Edgo Mi	tations	Use-case	LJ		T	W	T	M	FR		
# Euge Mit	lations		Н	S	Н	S	Н	S	Н	S	
1k			0.0056	0.0054	0.0186	0.016	0.0182	0.0179	0.0311	0.031	
10 <i>k</i>		SSSP	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	
1k			0.004	0.0036	0.0123	0.0123	0.0148	0.0174	0.0216	0.0229	
10 <i>k</i>		CC	0.006	0.006	0.0185	0.018	0.0224	0.0228	0.0348	0.0387	
100/	ć		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)

Anon.

PAGERANK (PR) I := $\lambda v. 1 / |V|$ Р $\lambda n, e. n / \text{outdeg}(\text{src}(e))$:= R $:= \lambda v, v', v + v'$ ε $:= \lambda n. \ \gamma * n + (1 - \gamma) / |V|$ $:= \lambda n, e. - \mathcal{E}^{-1}(n) / \text{outdeg}(\text{src}(e))$ ${\mathcal B}$ 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.

4215 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.
- Xiaowei Zhu, Wentao Han, and Wenguang Chen. Gridgraph: Large-scale graph processing on a single machine using
 Level hierarchical partitioning. In 2015 {USENIX} Annual Technical Conference ({USENIX} {ATC} 15), pages 375–386,
 2015.

###