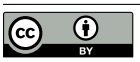
Verifying Graph Algorithms in Separation Logic: A Case for an Algebraic Approach (Appendices)

MARCOS GRANDURY, IMDEA Software Institute, Spain and Universidad Politécnica de Madrid, Spain
ALEKSANDAR NANEVSKI, IMDEA Software Institute, Spain
ALEXANDER GRYZLOV, IMDEA Software Institute, Spain

A Proof outline for the length-calculating program

$\{list\ \alpha_0\ (i, \text{null})\}$
$n := 0;$
$\{list\ \alpha_0\ (i, \text{null}) \wedge n = 0\}$
$j := i;$
$\{i = j \wedge list\ \alpha_0\ (i, \text{null}) \wedge n = 0\}$
$\{list\ []\ (i, j) * list\ \alpha_0\ (j, \text{null}) \wedge n = \#[] \wedge \alpha_0 = [] \bullet \alpha_0\}$
$\{\exists \alpha\ \beta.\ list\ \alpha\ (i, j) * list\ \beta\ (j, \text{null}) \wedge n = \#\alpha \wedge \alpha_0 = \alpha \bullet \beta\}$
while $j \neq \text{null}$ **do**
  $\{\exists \alpha\ \beta.\ list\ \alpha\ (i, j) * list\ \beta\ (j, \text{null}) \wedge n = \#\alpha \wedge \alpha_0 = \alpha \bullet \beta \wedge j \neq \text{null}\}$
  $\{\exists \alpha\ b\ \beta'.\ list\ \alpha\ (i, j) * list\ (b \bullet \beta')\ (j, \text{null}) \wedge n = \#\alpha \wedge \alpha_0 = \alpha \bullet (b \bullet \beta')\}$
  $\{\exists \alpha\ b\ \beta'\ k.\ list\ \alpha\ (i, j) * j \Longmapsto b, k * list\ \beta'\ (k, \text{null}) \wedge n = \#\alpha \wedge \alpha_0 = \alpha \bullet (b \bullet \beta')\}$
  $j := j.\text{next};$
  $\{\exists \alpha\ b\ \beta'\ j'.\ list\ \alpha\ (i, j') * j' \Longmapsto b, j * list\ \beta'\ (j, \text{null}) \wedge n = \#\alpha \wedge \alpha_0 = \alpha \bullet (b \bullet \beta')\}$
  $\{\exists \alpha\ b\ \beta'.\ list\ (\alpha \bullet b)\ (i, j) * list\ \beta'\ (j, \text{null}) \wedge n = \#\alpha \wedge \alpha_0 = (\alpha \bullet b) \bullet \beta'\}$
  $n := n + 1;$
  $\{\exists \alpha\ b\ \beta'.\ list\ (\alpha \bullet b)\ (i, j) * list\ \beta'\ (j, \text{null}) \wedge n = \#\alpha + 1 \wedge \alpha_0 = (\alpha \bullet b) \bullet \beta'\}$
  $\{\exists \alpha\ b\ \beta'.\ list\ (\alpha \bullet b)\ (i, j) * list\ \beta'\ (j, \text{null}) \wedge n = \#(\alpha \bullet b) \wedge \alpha_0 = (\alpha \bullet b) \bullet \beta'\}$
  $\{\exists \alpha'\ \beta'.\ list\ \alpha'\ (i, j) * list\ \beta'\ (j, \text{null}) \wedge n = \#\alpha' \wedge \alpha_0 = \alpha' \bullet \beta'\}$
end while
$\{\exists \alpha'\ \beta'.\ list\ \alpha'\ (i, j) * list\ \beta'\ (j, \text{null}) \wedge n = \#\alpha' \wedge \alpha_0 = \alpha' \bullet \beta' \wedge j = \text{null}\}$
$\{\exists \alpha'\ \beta'.\ list\ \alpha'\ (i, j) * list\ \beta'\ (j, \text{null}) \wedge n = \#\alpha' \wedge \alpha_0 = \alpha' \bullet \beta' \wedge j = \text{null} \wedge \beta' = []\}$
$\{\exists \alpha'.\ list\ \alpha'\ (i, \text{null}) * list\ []\ (\text{null}, \text{null}) \wedge n = \#\alpha' \wedge \alpha_0 = \alpha'\}$
$\{list\ \alpha_0\ (i, \text{null}) * emp \wedge n = \#\alpha_0\}$
$\{list\ \alpha_0\ (i, \text{null}) \wedge n = \#\alpha_0\}$

Authors' Contact Information: Marcos Grandury, IMDEA Software Institute, Madrid, Spain and Universidad Politécnica de Madrid, Madrid, Spain, marcos.grandury@imdea.org; Aleksandar Nanevski, IMDEA Software Institute, Madrid, Spain, aleks.nanevski@imdea.org; Alexander Gryzlov, IMDEA Software Institute, Madrid, Spain, aliaksandr.hryzlou@imdea.org.



B Proof outline for computing if t is marked (or null)

$\{\exists\gamma.\ graph\ \gamma \wedge inv\ \gamma_0\ \gamma\ t\ p\}$
$\{graph\ \gamma \wedge inv\ \gamma_0\ \gamma\ t\ p\}$
if $t = \text{null}$ **then**
$\{graph\ \gamma \wedge inv\ \gamma_0\ \gamma\ t\ p \wedge t = null\}$
$tm := \text{true}$
$\{graph\ \gamma \wedge inv\ \gamma_0\ \gamma\ t\ p \wedge t = null \wedge tm = \text{true}\}$
$\{graph\ \gamma \wedge inv\ \gamma_0\ \gamma\ t\ p \wedge tm = (t = null)\}$
else
$\{graph\ \gamma \wedge inv\ \gamma_0\ \gamma\ t\ p \wedge t \neq null\}$
$\{\exists m.\ t \Longmapsto m, -, - * graph\ \gamma\backslash t \wedge inv\ \gamma_0\ \gamma\ t\ p\}$
$tmp := t.m;$
$\{\exists m.\ t \Longmapsto m, -, - * graph\ \gamma\backslash t \wedge inv\ \gamma_0\ \gamma\ t\ p \wedge tmp = m\}$
$tm := (tmp \neq O)$
$\{\exists m.\ t \Longmapsto m, -, - * graph\ \gamma\backslash t \wedge inv\ \gamma_0\ \gamma\ t\ p \wedge tmp = m \wedge tm = (tmp \neq O)\}$
$\{\exists m.\ t \Longmapsto m, -, - * graph\ \gamma\backslash t \wedge inv\ \gamma_0\ \gamma\ t\ p \wedge tm = (m \neq O)\}$
$\{graph\ \gamma \wedge inv\ \gamma_0\ \gamma\ t\ p \wedge tm = (t.m \neq O)\}$
$\{graph\ \gamma \wedge inv\ \gamma_0\ \gamma\ t\ p \wedge tm = (t \in nodes\ \gamma/_{L,R,X})\}$
end if
$\{graph\ \gamma \wedge inv\ \gamma_0\ \gamma\ t\ p \wedge tm = (t = \text{null} \vee t \in nodes\ \gamma/_{L,R,X})\}$
$\{graph\ \gamma \wedge inv\ \gamma_0\ \gamma\ t\ p \wedge tm = (t \in marked_0\ \gamma)\}$
$\{\exists\gamma.\ graph\ \gamma \wedge inv\ \gamma_0\ \gamma\ t\ p \wedge tm = (t \in marked_0\ \gamma)\}$

C Proof for SWING

The pre- and postcondition for SWING derive from lines 18 and 20 of Fig. 10.

$$\begin{array}{c}\{\exists\gamma.\ graph\ \gamma \wedge inv\ \gamma_0\ \gamma\ t\ p \wedge t \in marked_0\ \gamma \wedge \gamma_{\text{val}}\ p = L\}\\ \text{SWING}\\ \{\exists\gamma'.\ graph\ \gamma' \wedge inv\ \gamma_0\ \gamma'\ t\ p\}\end{array} \tag{25}$$

The precondition says that the heap implements a well-formed graph γ, that satisfies the invariant. Additionally, t is marked or null and p is marked L. The postcondition asserts that the heap represents a new graph γ' that satisfies the invariant for the updated values of t and p.

1. $\{\exists\gamma.\ graph\ \gamma \wedge inv\ \gamma_0\ \gamma\ t\ p \wedge t \in marked_0\ \gamma \wedge \gamma_{\text{val}}\ p = L\}$
2. $\quad\{graph\ \gamma \wedge t = t_0 \wedge p = p_0$
   $\wedge inv\ \gamma_0\ \gamma\ t_0\ p_0 \wedge t_0 \in marked_0\ \gamma \wedge \gamma\ p_0 = (L, [p_l, p_r])\}$
3. $\quad\quad\{graph\ (p_0 \mapsto (L, [p_l, p_r]) \bullet \gamma\backslash p_0) \wedge t = t_0 \wedge p = p_0\}$
4. $\quad\quad\{(p_0 \Longmapsto L, p_l, p_r \wedge t = t_0 \wedge p = p_0) * graph\ \gamma\backslash p_0\}$
5. $\quad\quad\quad\{p_0 \Longmapsto L, p_l, p_r \wedge t = t_0 \wedge p = p_0\}$
   $tmp_1 := p.r;\ tmp_2 := p.l;\ p.r := tmp_2;\ p.l := t;\ p.m := R;\ t := tmp_1;$

6. $\{p_0 \Mapsto R, t_0, p_l \wedge t = p_r \wedge p = p_0\}$
7. $\{(p_0 \Mapsto R, t_0, p_l \wedge t = p_r \wedge p = p_0) * graph\ \gamma\backslash p_0\}$
8. $\{graph\ (p_0 \mapsto (R, [t_0, p_l]) \bullet \gamma\backslash p_0) \wedge t = p_r \wedge p = p_0\}$
9. $\{graph\ (p_0 \mapsto (R, [t_0, p_l]) \bullet \gamma\backslash p_0) \wedge t = p_r \wedge p = p_0$
   $\wedge inv\ \gamma_0\ \gamma\ t_0\ p_0 \wedge t_0 \in marked_0\ \gamma \wedge \gamma\ p_0 = (L, [p_l, p_r])\}$
10. $\{\exists \gamma'.\ graph\ \gamma' \wedge inv\ \gamma_0\ \gamma'\ t\ p\}$

Line 9 implies line 10. As for the case of POP, this step involves reformulating it as an implication, where initial and final values of the graph, stack and nodes t and p are made explicit.

$$\gamma = p_0 \mapsto (L, [p_l, p_r]) \bullet \gamma\backslash p_0 \wedge \tag{26}$$
$$inv'\ \gamma_0\ \gamma\ \alpha\ t_0\ p_0 \wedge \tag{27}$$
$$t_0 \in marked_0\ \gamma \implies \tag{28}$$
$$\exists \gamma'.\ \gamma' = p_0 \mapsto (R, [t_0, p_l]) \bullet \gamma\backslash p_0 \wedge \tag{29}$$
$$inv'\ \gamma_0\ \gamma'\ \alpha\ p_r\ p_0 \tag{30}$$

SWING differs from the other two operations in that the stack remains unaltered. Furthermore, because we know $p_0 \neq \text{null}$, $uniq\ (\text{null} \bullet \alpha)$ and $p_0 = last\ (\text{null} \bullet \alpha)$ it follows that $\alpha = \alpha' \bullet p_0$ for some sequence α'. The invariant (*a*) $p_r \in nodes_0\ \gamma'$ follows from (26) and (27) as we know that $p_r \in sinks\ \gamma$, $closed\ \gamma$ and $nodes_0\ \gamma = nodes_0\ \gamma'$.

(*b*) $sinks\ \gamma' =$	Def. of γ'
$= sinks\ (p_0 \mapsto (R, [t_0, p_l]) \bullet \gamma\backslash p_0)$	Lem. 3.1 (6) (distrib.)
$= sinks\ (p_0 \mapsto (R, [t_0, p_l])) \cup sinks\ (\gamma\backslash p_0)$	Def. of *sinks* (11)
$= \{t_0, p_l\} \cup sinks\ (\gamma\backslash p_0)$	Set inclusion
$\subseteq \{t_0, p_l, p_r\} \cup sinks\ (\gamma\backslash p_0)$	Assump. (27) & (28)
$\subseteq nodes_0\ \gamma$	Def. of *nodes*
$= nodes_0\ \gamma'$	

(*c*) $nodes\ \gamma'/_{L,R} =$	Def. of γ'
$= nodes\ (p_0 \mapsto (R, [t_0, p_l]) \bullet \gamma\backslash p_0)/_{L,R}$	Lem. 3.1 (1) (distrib.)
$= nodes\ (p_0 \mapsto (R, [t_0, p_l]))/_{L,R} \cup nodes\ (\gamma\backslash p_0)/_{L,R}$	Def. of *filter* (10)
$= nodes\ (p_0 \mapsto (R, [t_0, p_l]))\ \cup nodes\ (\gamma\backslash p_0)/_{L,R}$	Def. of *nodes*
$= \{p_0\} \cup nodes\ (\gamma\backslash p_0)/_{L,R}$	Def. of *nodes*
$= nodes\ (p_0 \mapsto (L, [p_l, p_r]))\ \cup nodes\ (\gamma\backslash p_0)/_{L,R}$	Def. of *filter* (10)
$= nodes\ (p_0 \mapsto (L, [p_l, p_r]))/_{L,R} \cup nodes\ (\gamma\backslash p_0)/_{L,R}$	Lem. 3.1 (1) (distrib.)
$= nodes\ (p_0 \mapsto (L, [p_l, p_r]) \bullet \gamma\backslash p_0)/_{L,R}$	Assump. (27)
$= \alpha$	

(*d*) $inset\ \alpha\ \gamma' =$	Def. of γ'
$= inset\ \alpha\ (p_0 \mapsto (R, [t_0, p_l]) \bullet \gamma\backslash p_0)$	Lem. 3.1 (5) (distrib.)
$= inset\ \alpha\ (p_0 \mapsto (R, [t_0, p_l])) \bullet inset\ \alpha\ \gamma\backslash p_0$	Def. of *inset* (14)

$= p_0 \mapsto [t_0, \mathit{prev}\ (\mathrm{null} \bullet \alpha)\ p_0] \bullet \mathit{inset}\ \alpha\ \gamma\backslash p_0$ — Assump. (27)
$= p_0 \mapsto [t_0, p_l] \bullet |\gamma\backslash p_0|$ — Def. of *erasure* (8)
$= |p_0 \mapsto (R, [t_0, p_l])| \bullet |\gamma\backslash p_0|$ — Lem. 3.1 (4) (distrib.)
$= |p_0 \mapsto (R, [t_0, p_l]) \bullet \gamma\backslash p_0|$ — Def. of γ'
$= |\gamma'|$

(e) $\mathit{restore}\ \alpha\ p_r\ \gamma' =$ — Def. of γ'
$= \mathit{restore}\ \alpha\ p_r\ (p_0 \mapsto (R, [t_0, p_l]) \bullet \gamma\backslash p_0)$ — Lem. 3.1 (5) (distrib.)
$= \mathit{restore}\ \alpha\ p_r\ (p_0 \mapsto (R, [t_0, p_l])) \bullet \mathit{restore}\ \alpha\ p_r\ \gamma\backslash p_0$ — Def. of *restore* (15)
$= p_0 \mapsto [t_0, \mathit{next}\ (\alpha \bullet p_r)\ p_0] \bullet \mathit{restore}\ \alpha\ p_r\ \gamma\backslash p_0$ — Assump. (27)
$= p_0 \mapsto [t_0, p_r] \bullet \mathit{restore}\ \alpha\ p_r\ \gamma\backslash p_0$ — Lem. 17
$= p_0 \mapsto [t_0, p_r] \bullet \mathit{restore}\ \alpha\ t_0\ \gamma\backslash p_0$ — Def. of *restore* (15)
$= \mathit{restore}\ \alpha\ t_0\ (p_0 \mapsto (L, [p_l, p_r])) \bullet \ \mathit{restore}\ \alpha\ t_0\ \gamma\backslash p_0$ — Lem. 3.1 (5) (distrib.)
$= \mathit{restore}\ \alpha\ t_0\ (p_0 \mapsto (L, [p_l, p_r]) \bullet \gamma\backslash p_0)$ — Def. of γ
$= \mathit{restore}\ \alpha\ t_0\ \gamma$ — Assump. (27)
$= |\gamma_0|$

(f) $\mathit{nodes}\ \gamma'/_O =$ — Def. of γ'
$= \mathit{nodes}\ (p_0 \mapsto (R, [t_0, p_l]) \bullet \gamma\backslash p_0)/_O$ — Lem. 3.1 (3) (distrib.)
$= \mathit{nodes}\ ((p_0 \mapsto (R, [t_0, p_l]))/_O \bullet (\gamma\backslash p_0)/_O)$ — Lem. 3.1 (1) (distrib.)
$= \mathit{nodes}\ (p_0 \mapsto (R, [t_0, p_l]))/_O \cup \mathit{nodes}\ (\gamma\backslash p_0)/_O$ — Def. of *filter* (10)
$= \mathit{nodes}\ e \cup \mathit{nodes}\ (\gamma\backslash p_0)/_O$ — Def. of *filter* (10)
$= \mathit{nodes}\ (p_0 \mapsto (L, [p_l, p_r]))/_O \cup \mathit{nodes}\ (\gamma\backslash p_0)/_O$ — Lem. 3.1 (1) (distrib.)
$= \mathit{nodes}\ ((p_0 \mapsto (L, [p_l, p_r]))/_O \bullet (\gamma\backslash p_0)/_O)$ — Lem. 3.1 (3) (distrib.)
$= \mathit{nodes}\ (p_0 \mapsto (L, [p_l, p_r]) \bullet \gamma\backslash p_0)/_O$ — Def. of γ
$= \mathit{nodes}\ \gamma/_O$ — Assump. (27)
$\subseteq \bigcup_{\alpha' \cdot p_0} (\mathit{reach}\ \gamma/_O \circ \gamma_r) \cup \mathit{reach}\ \gamma/_O\ t_0$ — $t_0 \notin \mathit{nodes}\ \gamma/_O$
$= \bigcup_{\alpha' \cdot p_0} (\mathit{reach}\ \gamma/_O \circ \gamma_r)$ — Comm.&Assoc. of $\cup$
$= \bigcup_{\alpha'} (\mathit{reach}\ \gamma/_O \circ \gamma_r) \cup \mathit{reach}\ \gamma/_O\ p_r$ — $\gamma/_O = \gamma'/_O$
$= \bigcup_{\alpha'} (\mathit{reach}\ \gamma'/_O \circ \gamma_r) \cup \mathit{reach}\ \gamma'/_O\ p_r$ — $\gamma_r = \gamma'_r$ on α'
$= \bigcup_{\alpha'} (\mathit{reach}\ \gamma'/_O \circ \gamma'_r) \cup \mathit{reach}\ \gamma'/_O\ p_r$ — $\gamma'_r\ p_0 \notin \mathit{nodes}\ \gamma'/_O$
$= \bigcup_{\alpha' \cdot p_0} (\mathit{reach}\ \gamma'/_O \circ \gamma'_r) \cup \mathit{reach}\ \gamma'/_O\ p_r$

D Proof for PUSH

The pre- and postcondition for PUSH derive from lines 23 and 25 of Fig. 10.

$$\begin{array}{c}\{\exists\gamma.\ graph\ \gamma \wedge inv\ \gamma_0\ \gamma\ t\ p \wedge t \notin marked_0\ \gamma\}\\ \text{PUSH}\\ \{\exists\gamma'.\ graph\ \gamma' \wedge inv\ \gamma_0\ \gamma'\ t\ p\}\end{array} \tag{31}$$

The precondition says that the heap implements a well-formed graph γ, that satisfies the invariant and t is an unmarked node different from null. The postcondition asserts that the heap represents a new graph γ' that satisfies the invariant for the updated values of t and p.

1. $\{\exists\gamma.\ graph\ \gamma \wedge inv\ \gamma_0\ \gamma\ t\ p \wedge t \notin marked_0\ \gamma\}$
2. $\quad\{graph\ \gamma \wedge t = t_0 \wedge p = p_0$
   $\quad\wedge inv\ \gamma_0\ \gamma\ t_0\ p_0 \wedge \gamma\ t_0 = (O, [t_l, t_r])\}$
3. $\quad\quad\{graph\ (t_0 \mapsto (O, [t_l, t_r]) \bullet \gamma\backslash t_0) \wedge t = t_0 \wedge p = p_0\}$
4. $\quad\quad\{(t_0 \Longmapsto O, t_l, t_r \wedge t = t_0 \wedge p = p_0) * graph\ \gamma\backslash t_0\}$
5. $\quad\quad\quad\{t_0 \Longmapsto O, t_l, t_r \wedge t = t_0 \wedge p = p_0\}$
   $\quad\quad\quad tmp := t.l;\ t.l := p;\ t.m := L;\ p := t;\ t := tmp;$
6. $\quad\quad\quad\{t_0 \Longmapsto L, p_0, t_r \wedge t = t_l \wedge p = t_0\}$
7. $\quad\quad\{(t_0 \Longmapsto L, p_0, t_r \wedge t = t_l \wedge p = t_0) * graph\ \gamma\backslash t_0\}$
8. $\quad\quad\{graph\ (t_0 \mapsto (L, [p_0, t_r]) \bullet \gamma\backslash t_0) \wedge t = t_l \wedge p = t_0\}$
9. $\quad\{graph\ (t_0 \mapsto (L, [p_0, t_r]) \bullet \gamma\backslash t_0) \wedge t = t_l \wedge p = t_0$
   $\quad\wedge inv\ \gamma_0\ \gamma\ t_0\ p_0 \wedge \gamma\ t_0 = (O, [t_l, t_r])\}$
10. $\{\exists\gamma'.\ graph\ \gamma' \wedge inv\ \gamma_0\ \gamma'\ t\ p\}$

Line 9 implies line 10. Proving this last step corresponds to proving the following implication.

$$\gamma = t_0 \mapsto (O, [t_l, t_r]) \bullet \gamma\backslash t_0 \wedge \tag{32}$$
$$inv'\ \gamma_0\ \gamma\ \alpha\ t_0\ p_0 \implies \tag{33}$$
$$\exists\gamma'.\ \gamma' = t_0 \mapsto (L, [p_0, t_r]) \bullet \gamma\backslash t_0 \wedge \tag{34}$$
$$inv'\ \gamma_0\ \gamma'\ (\alpha \bullet t_0)\ t_l\ t_0 \tag{35}$$

Proving invariant (*a*) requires showing that (32) and (33) imply $uniq\ (\text{null} \bullet \alpha \bullet t_0)$, $t_0 = last\ (\text{null} \bullet \alpha \bullet t_0)$ and $t_l \in nodes_0\ \gamma'$. From $\gamma_m\ t_0 = O$ and $nodes\ \gamma/_{L,R} = \alpha$ it follows $uniq\ (\text{null} \bullet \alpha \bullet t_0)$. By 32 it follows $t_l \in sinks\ \gamma$. Then $closed\ \gamma$ implies that $t_l \in nodes_0\ \gamma$. And finally $nodes_0\ \gamma = nodes_0\ \gamma'$ so $t_l \in nodes_0\ \gamma'$.

(*b*) $sinks\ \gamma' =$	Def. of γ'
$= sinks\ (t_0 \mapsto (L, [p_0, t_r]) \bullet \gamma\backslash t_0)$	Lem. 3.1 (6) (distrib.)
$= sinks\ (t_0 \mapsto (L, [p_0, t_r])) \cup sinks\ (\gamma\backslash t_0)$	Def. of *sinks* (11)
$= \{p_0, t_r\} \cup sinks\ (\gamma\backslash t_0)$	Set inclusion
$\subseteq \{p_0, t_l, t_r\} \cup sinks\ (\gamma\backslash t_0)$	Assump. (33)
$\subseteq nodes_0\ \gamma$	Def. of *nodes*
$= nodes_0\ \gamma'$	

$(c)\ \mathit{nodes}\ \gamma'/_{L,R} =$	Def. of γ'
$= \mathit{nodes}\ (t_0 \mapsto (L, [p_0, t_r]) \bullet \gamma\backslash t_0)/_{L,R}$	Lem. 3.1 (3) (distrib.)
$= \mathit{nodes}\ ((t_0 \mapsto (L, [p_0, t_r]))/_{L,R} \bullet (\gamma\backslash t_0)/_{L,R})$	Def. of *filter* (10)
$= \mathit{nodes}\ (t_0 \mapsto (L, [p_0, t_r]) \bullet (\gamma\backslash t_0)/_{L,R})$	Lem. 3.1 (1) (distrib.)
$= \mathit{nodes}\ (t_0 \mapsto (L, [p_0, t_r])) \cup \mathit{nodes}\ (\gamma\backslash t_0)/_{L,R}$	Def. of *nodes*
$= \{t_0\} \cup \mathit{nodes}\ (\gamma\backslash t_0)/_{L,R}$	Def. of *filter* (10)
$= \{t_0\} \cup \mathit{nodes}\ (t_0 \mapsto (O, [t_l, t_r]) \bullet \gamma\backslash t_0)/_{L,R}$	Def. of γ
$= \{t_0\} \cup \mathit{nodes}\ \gamma/_{L,R}$	Assump. (33)
$= \{t_0\} \cup \alpha$	Set equality
$= (\alpha \bullet t_0)$	

$(d)\ \mathit{inset}\ (\alpha \bullet t_0)\ \gamma' =$	Def. of γ'
$= \mathit{inset}\ (\alpha \bullet t_0)\ (t_0 \mapsto (L, [p_0, t_r]) \bullet \gamma\backslash t_0)$	Lem. 3.1 (5) (distrib.)
$= \mathit{inset}\ (\alpha \bullet t_0)\ (t_0 \mapsto (L, [p_0, t_r])) \bullet \mathit{inset}\ (\alpha \bullet t_0)\ \gamma\backslash t_0$	Def. of *inset* (14)
$= t_0 \mapsto [\mathit{prev}\ (\mathrm{null} \bullet \alpha \bullet t_0)\ t_0, t_r] \bullet \mathit{inset}\ (\alpha \bullet t_0)\ \gamma\backslash t_0$	Assump. (33)
$= t_0 \mapsto [p_0, t_r] \bullet \mathit{inset}\ (\alpha \bullet t_0)\ \gamma\backslash t_0$	Lem. 16
$= t_0 \mapsto [p_0, t_r] \bullet \mathit{inset}\ \alpha\ \gamma\backslash t_0$	Def. of *erasure* (8)
$= \lvert t_0 \mapsto (L, [p_0, t_r])\rvert \bullet \lvert\gamma\backslash t_0\rvert$	Lem. 3.1 (4) (distrib.)
$= \lvert t_0 \mapsto (L, [p_0, t_r]) \bullet \gamma\backslash t_0\rvert$	Def. of γ'
$= \lvert\gamma'\rvert$	

$(e)\ \mathit{restore}\ (\alpha \bullet t_0)\ t_l\ \gamma' =$	Def. of γ'
$= \mathit{restore}\ (\alpha \bullet t_0)\ t_l\ (t_0 \mapsto (L, [p_0, t_r]) \bullet \gamma\backslash t_0)$	Lem. 3.1 (5) (distrib.)
$= \mathit{restore}\ (\alpha \bullet t_0)\ t_l\ (t_0 \mapsto (L, [p_0, t_r]))$ $\bullet\ \mathit{restore}\ (\alpha \bullet t_0)\ t_l\ \gamma\backslash t_0$	Def. of *restore* (15)
$= t_0 \mapsto [\mathit{next}\ (\alpha \bullet t_0 \bullet t_l)\ t_0, t_r] \bullet \mathit{restore}\ (\alpha \bullet t_0)\ t_l\ \gamma\backslash t_0$	Assump. (33)
$= t_0 \mapsto [t_l, t_r] \bullet \mathit{restore}\ (\alpha \bullet t_0)\ t_l\ \gamma\backslash t_0$	Lem. 17
$= t_0 \mapsto [t_l, t_r] \bullet \mathit{restore}\ \alpha\ t_0\ \gamma\backslash t_0$	Def. of *restore* (15)
$= \mathit{restore}\ \alpha\ t_0\ (t_0 \mapsto (O, [t_l, t_r]))$ $\bullet\ \mathit{restore}\ \alpha\ t_0\ \gamma\backslash t_0$	Lem. 3.1 (5) (distrib.)
$= \mathit{restore}\ \alpha\ t_0\ (t_0 \mapsto (O, [t_l, t_r]) \bullet \gamma\backslash t_0)$	Def. of γ
$= \mathit{restore}\ \alpha\ t_0\ \gamma$	Assump. (33)
$= \lvert\gamma_0\rvert$	

$(f)\ \mathit{nodes}\ \gamma'/_O =$	Def. of γ'
$= \mathit{nodes}\ (t_0 \mapsto (L, [p_0, t_r]) \bullet \gamma\backslash t_0)/_O$	Lem. 3.1 (3) (distrib.)
$= \mathit{nodes}\ ((t_0 \mapsto (L, [p_0, t_r]))/_O \bullet (\gamma\backslash t_0)/_O)$	Lem. 3.1 (1) (distrib.)
$= \mathit{nodes}\ (t_0 \mapsto (L, [p_0, t_r]))/_O \cup \mathit{nodes}\ (\gamma\backslash t_0)/_O$	Def. of *filter* (10)
$= \mathit{nodes}\ e \cup \mathit{nodes}\ (\gamma\backslash t_0)/_O$	Def. of *nodes*

$$
\begin{aligned}
&= \mathit{nodes}\ (\gamma \backslash t_0)/_O && \text{Set difference}\\
&= (\{t_0\} \cup \mathit{nodes}\ (\gamma \backslash t_0)/_O) \backslash t_0 && \text{Def. of } \mathit{nodes}\\
&= (\mathit{nodes}\ (t_0 \mapsto (O, [t_l, t_r])) \cup \mathit{nodes}\ (\gamma \backslash t_0)/_O) \backslash t_0 && \text{Def. of } \mathit{filter}\ (10)\\
&= (\mathit{nodes}\ (t_0 \mapsto (O, [t_l, t_r]))/_O \cup \mathit{nodes}\ (\gamma \backslash t_0)/_O) \backslash t_0 && \text{Lem. 3.1 (1) (distrib.)}\\
&= (\mathit{nodes}\ (t_0 \mapsto (O, [t_l, t_r]) \bullet \gamma \backslash t_0)/_O) \backslash t_0 && \text{Def. of } \gamma\\
&= (\mathit{nodes}\ \gamma/_O) \backslash t_0 && \text{Assump. (33)}\\
&\subseteq (\bigcup_\alpha (\mathit{reach}\ \gamma/_O \circ \gamma_r) \cup \mathit{reach}\ \gamma/_O\ t_0) \backslash t_0 && \text{Lem. 3.4 (2) \& 3.4 (3)}\\
&= (\bigcup_\alpha (\mathit{reach}\ (\gamma/_O) \backslash t_0 \circ \gamma_r) \cup \mathit{reach}\ \gamma/_O\ t_0) \backslash t_0 && \text{Def. of } \mathit{reach}\ (13)\\
&= (\bigcup_\alpha (\mathit{reach}\ (\gamma/_O) \backslash t_0 \circ \gamma_r) \cup \{t_0\} \cup \bigcup_{\{t_l, t_r\}} \mathit{reach}\ (\gamma/_O)) \backslash t_0 && \text{Set difference}\\
&= \bigcup_\alpha (\mathit{reach}\ (\gamma/_O) \backslash t_0 \circ \gamma_r) \cup \bigcup_{\{t_l, t_r\}} \mathit{reach}\ (\gamma/_O) \backslash t_0 && (\gamma/_O) \backslash t_0 = \gamma'/_O\\
&= \bigcup_\alpha (\mathit{reach}\ \gamma'/_O) \circ \gamma_r) \cup \bigcup_{\{t_l, t_r\}} \mathit{reach}\ \gamma'/_O && \gamma_r = \gamma'_r \text{ on } (\alpha \bullet t_0)\\
&= \bigcup_{(\alpha \bullet t_0)} (\mathit{reach}\ \gamma'/_O \circ \gamma'_r) \cup \mathit{reach}\ \gamma'/_O\ t_l
\end{aligned}
$$

E Union-Find Data Structure

E.1 Non-spatial definitions

$$
\begin{aligned}
\mathit{cycles}\ \gamma &\mathrel{\widehat{=}} \{x \mid x \in \bigcup_{y \in \gamma_{\text{adj}}\ x} \mathit{reach}\ \gamma\ y\}\\
\mathit{preacyclic}\ \gamma &\mathrel{\widehat{=}} \mathit{cycles}\ \gamma \subseteq \mathit{loops}\ \gamma\\
\mathit{dangls}\ \gamma &\mathrel{\widehat{=}} (\mathit{sinks}\ \gamma) \backslash \mathit{nodes}\ \gamma
\end{aligned}
$$

The function *cycles* computes the set of nodes that constitute a cycle in the graph. More precisely, it identifies nodes that are reachable from one of their adjacent nodes. This avoids the trivial case where every node is reachable from itself. *Loops* are cycles of size one—that is, nodes that explicitly include themselves in their adjacency list. Then a *preacyclic* graph is one where every cycle is of size one. Loops are given a special status as they are cycles that do not break under decomposition. *dangls* γ selects the dangling nodes of a graph, those that are pointed at by a node in the graph but are not themselves in the graph. Notice that null may be in this set.

Lemma E.1 (Union-find abstractions).

(1) $\mathit{dangls}\ (\gamma_1 \bullet \gamma_2) = (\mathit{dangls}\ \gamma_1) \backslash \mathit{nodes}\ \gamma_2 \cup (\mathit{dangls}\ \gamma_2) \backslash \mathit{nodes}\ \gamma_1$
(2) $\mathit{loops}\ \gamma \subseteq \mathit{cycles}\ \gamma \subseteq \mathit{nodes}\ \gamma$
(3) $\mathit{dangls}\ \gamma \cap \mathit{nodes}\ \gamma = \emptyset$

Lemma E.2 (Summit characterization). *Let γ be a graph and $x \in \mathit{nodes}\ \gamma$. Then, $z \in \mathit{summit}\ \gamma\ x$ iff there exists a path from x to y in γ such that z is a child of y, and either $z \notin \mathit{nodes}\ \gamma$ (i.e., the edge from y to z is dangling) or z is already in the path from x to y.*

Lemma E.3 (Summits).

(1) $\mathit{summits}\ \gamma = \mathit{cycles}\ \gamma \cup \mathit{dangls}\ \gamma$
(2) *If* $\mathit{summits}\ \gamma_1 = \mathit{summits}\ \gamma_2$ *then* $\mathit{summits}\ (\gamma_1 \bullet \gamma_2) = \mathit{summits}\ \gamma_1$
(3) *If* $x \in \mathit{loops}\ \gamma$ *then* $\mathit{summits}\ \gamma \backslash x \subseteq \mathit{summits}\ \gamma$

Lemma E.4 (Summits of inverted forest). *Let $\gamma = (\gamma_1 \bullet \gamma_2)$ be an an inverted forest ($\mathit{summits}\ \gamma \subseteq \mathit{loops}\ \gamma$) then*

(1) $sinks\ \gamma \subseteq nodes\ \gamma\ (closed\ \gamma)$
(2) $cycles\ \gamma \subseteq loops\ \gamma\ (preacyclic\ \gamma)$
(3) $summits\ (\gamma_1 \bullet \gamma_2) = (summits\ \gamma_1) \backslash nodes\ \gamma_2 \cup (summits\ \gamma_2) \backslash nodes\ \gamma_1$
(4) $summit\ \gamma\ x = summit\ \gamma\ (\gamma\ x)$

Lemma E.5 (Preacyclic mutation). *Let γ be a unary graph such that preacyclic γ, and $x \in nodes\ \gamma$ and $y \notin nodes\ \gamma$. The graph γ' obtained by modifying x's successor to y, also satisfies preacyclic γ'.*

E.2 Proof outline for NEW

1. $\{emp\}$
2. $p := alloc\ \text{null};$
3. $\{p \Longmapsto \text{null}\}$
4. $p.next := p;$
5. $\{p \Longmapsto p\}$
6. **return** p
7. $\{p \Longmapsto p \wedge result = p\}$
8. $\{graph_1\ (p \mapsto p) \wedge summits\ (p \mapsto p) = loops\ (p \mapsto p) = nodes\ (p \mapsto p) = \{p\} \wedge result = p\}$
9. $\{\exists \gamma.\ graph_1\ \gamma \wedge summits\ \gamma = loops\ \gamma = \{p\} \wedge nodes\ \gamma = \{p\} \wedge result = p\}$
10. $\{set\ \{result\}\ result\}$

The first two commands are standard separation logic, the third command in line 6 corresponds to a value returning function supported by Hoare Type Theory. The step from line 7 to line 8 lifts the reasoning from the heap to the abstract graph and establishes that the singleton graph is indeed an inverted tree with *result* as its representative. Finally the conjunct are folded into the set predicate.

E.3 Proof outline for FIND

1. $\{set\ S\ y \wedge x \in S\}$
2. $\{\exists \gamma.\ graph_1\ \gamma \wedge summits\ \gamma = loops\ \gamma = \{y\} \wedge nodes\ \gamma = S \wedge x \in S\}$
3. $\quad\{graph_1\ \gamma \wedge summits\ \gamma = loops\ \gamma = \{y\} \wedge nodes\ \gamma = S \wedge x \in S\}$
4. $\quad\{graph_1\ (x \mapsto \gamma\ x \bullet \gamma \backslash x) \wedge x \in S \wedge\ summits\ \gamma = loops\ \gamma = \{y\} \wedge nodes\ \gamma = S\}$
5. $\qquad\{x \Longmapsto \gamma\ x * graph_1\ \gamma \backslash x \wedge x \in S\}$
6. $\qquad p := x.next;$
7. $\qquad\{x \Longmapsto \gamma\ x * graph_1\ \gamma \backslash x \wedge x \in S \wedge p = \gamma\ x\}$
8. $\qquad\{graph_1\ (x \mapsto \gamma\ x \bullet \gamma \backslash x) \wedge x \in S \wedge p = \gamma\ x\}$
9. $\qquad\{graph_1\ \gamma \wedge x \in S \wedge p = \gamma\ x\}$
10. $\qquad$**while** $p \neq x$ **do**
11. $\qquad\quad\{graph_1\ \gamma \wedge x \in S \wedge p = \gamma\ x \neq x\}$
12. $\qquad\quad x := p;$
13. $\qquad\quad\{graph_1\ \gamma \wedge x \in S \wedge x = p\}$
14. $\qquad\quad\{graph_1\ (x \mapsto \gamma\ x \bullet \gamma \backslash x) \wedge x \in S \wedge x = p\}$

15. $\{x \Mapsto \gamma\ x * graph_1\ \gamma \backslash x \wedge x \in S \wedge x = p\}$
16. $p := x.next;$
17. $\{x \Mapsto \gamma\ x * graph_1\ \gamma \backslash x \wedge x \in S \wedge p = \gamma\ x\}$
18. $\{graph_1\ (x \mapsto nx \bullet \gamma \backslash x) \wedge x \in S \wedge p = \gamma\ x\}$
19. $\{graph_1\ \gamma \wedge x \in S \wedge p = \gamma\ x\}$
20. **end while**
21. $\{graph_1\ \gamma \wedge x \in S \wedge p = \gamma\ x = x\}$
22. **return** x
23. $\{graph_1\ \gamma \wedge x \in S \wedge p = \gamma\ x = x = result\}$
24. $\{graph_1\ \gamma \wedge result = y \wedge summits\ \gamma = loops\ \gamma = \{y\} \wedge nodes\ \gamma = S\}$
25. $\{\exists \gamma.\ graph_1\ \gamma \wedge summits\ \gamma = loops\ \gamma = \{y\} \wedge nodes\ \gamma = S \wedge result = y\}$
26. $\{set\ S\ y \wedge result = y\}$

The first five lines of the proof outline unfold the spatial predicates, first set, then *graph*, to reveal the node corresponding to the argument x. The non-spatial conjuncts involving γ and y are framed early on, as these values remain unchanged throughout the execution and will later be reintroduced without modification. The command at line 6 assigns to p the successor of x. In line 10, p is compared to x. A mismatch implies that the root node, which points to itself, has not yet been found, and thus the loop must be entered. Within the loop, the first command advances x to its parent node. Line 13 requires showing that the new value of x remains within S. This holds because x is guaranteed to be in $nodes\ \gamma$, given that $closed\ \gamma$ holds by lemma E.4 (1) and $nodes\ \gamma = S$. Next, p is updated to be the parent of the current x. By the end of the loop in line 19, its invariant from line 9 is reestablished. When the loop terminates, the invariant together with the negation of the loop guard leads to the conclusion in line 21: x is a node in S and points to itself in γ. Line 22 sets the result of the program to be x, which by the non-spatial conjunct framed at the beginning we know must be y, as it is established to be the only self-pointing node in γ.

E.4 Proof outline for UNION

1. $\{set\ S_1\ x_1 * set\ S_2\ x_2\}$
2. $\{graph_1\ \gamma_1 * graph_1\ \gamma_2$
3. $\wedge\ summits\ \gamma_1 = loops\ \gamma_1 = \{x_1\} \wedge nodes\ \gamma_1 = S_1$
4. $\wedge\ summits\ \gamma_2 = loops\ \gamma_2 = \{x_2\} \wedge nodes\ \gamma_2 = S_2\}$
5. $\{graph_1(x_1 \mapsto x_1 \bullet \gamma_1 \backslash x_1)\}$
6. $\{x_1 \Mapsto x_1 * graph_1(\gamma_1 \backslash x_1)\}$
7. $x_1.next := x_2;$
8. $\{x_1 \Mapsto x_2 * graph_1(\gamma_1 \backslash x_1)\}$
9. **return** x_2
10. $\{x_1 \Mapsto x_2 * graph_1(\gamma_1 \backslash x_1) \wedge result = x_2\}$
11. $\{graph_1(x_1 \mapsto x_2 \bullet \gamma_1 \backslash x_1) \wedge result = x_2\}$
12. $\{graph_1(x_1 \mapsto x_2 \bullet \gamma_1 \backslash x_1) \wedge result = x_2 * graph_1\ \gamma_2$

13. $\quad \wedge\ summits\ \gamma_1 = loops\ \gamma_1 = \{x_1\} \wedge nodes\ \gamma_1 = S_1$
14. $\quad \wedge\ summits\ \gamma_2 = loops\ \gamma_2 = \{x_2\} \wedge nodes\ \gamma_2 = S_2\}$
15. $\quad \{graph_1(x_1 \mapsto x_2 \bullet \gamma_1 \backslash x_1) * graph_1\ \gamma_2 \wedge result = x_2$
16. $\quad \wedge\ summits\ (x_1 \mapsto x_2 \bullet \gamma_1 \backslash x_1) = \{x_2\}$
17. $\quad \wedge\ loops\ (x_1 \mapsto x_2 \bullet \gamma_1 \backslash x_1) = \emptyset$
18. $\quad \wedge\ nodes\ (x_1 \mapsto x_2 \bullet \gamma_1 \backslash x_1) = S_1$
19. $\quad \wedge\ summits\ \gamma_2 = loops\ \gamma_2 = \{x_2\} \wedge nodes\ \gamma_2 = S_2\}$
20. $\quad \{graph_1(x_1 \mapsto x_2 \bullet \gamma_1 \backslash x_1 \bullet \gamma_2) \wedge result = x_2$
21. $\quad \wedge\ summits\ (x_1 \mapsto x_2 \bullet \gamma_1 \backslash x_1 \bullet \gamma_2) = \{x_2\}$
22. $\quad \wedge\ loops\ (x_1 \mapsto x_2 \bullet \gamma_1 \backslash x_1 \bullet \gamma_2) = \{x_2\}$
23. $\quad \wedge\ nodes\ (x_1 \mapsto x_2 \bullet \gamma_1 \backslash x_1 \bullet \gamma_2) = S_1 \bullet S_2\}$
24. $\{set\ (S_1 \bullet S_2)\ result \wedge result \in \{x_1, x_2\}\}$

Similar to the FIND proof, the initial lines of the proof outline (lines 1-6) unfold the spatial predicates isolating the node corresponding to the argument x_1. Line 2 unfolds the set predicate, instantiates the existentially quantified graphs as γ_1 and γ_2 and frames out $graph_1\ \gamma_2$ as well as the non-spatial facts about the initial disjoint graphs. Line 5 rewrites by the equality $\gamma_1 = x_1 \mapsto x_1 \bullet \gamma_1 \backslash x_1$ which follows from $x_1 \in loops\ \gamma_1$. The command in line 7 makes x_1 point to x_2. The second and final command sets x_2 as the result of the program. After the final command, in line 11 reasoning is lifted from the heap level and in line 12 the spatial conjuncts initially framed are reintroduced. In line 15 the abstractions that constitute the specification are recomputed for the mutated graph $x_1 \mapsto x_2 \bullet \gamma_1 \backslash x_1$, exploiting their distributivity. Before applying the distributivity of *summits*, preacyclicity is proved for the mutated graph by application of lemma E.5.

$summits\ (x_1 \mapsto x_2 \bullet \gamma_1 \backslash x_1) =$	
$= (summits\ (x_1 \mapsto x_2)) \backslash nodes\ (\gamma_1 \backslash x_1)$	Distrib. of *summits*
$\cup\ (summits\ \gamma_1 \backslash x_1) \backslash nodes\ (x_1 \mapsto x_2)$	
$= \{x_2\} \backslash nodes\ (\gamma_1 \backslash x_1)\ \cup\ (summits\ \gamma_1 \backslash x_1) \backslash x_1$	Def. of *summits* and *nodes*
$= \{x_2\}\ \cup\ (summits\ \gamma_1 \backslash x_1) \backslash x_1$	$x_2 \in nodes\ \gamma_2$ and $nodes\ \gamma_1 \backslash x_1 \cap nodes\ \gamma_2 = \emptyset$
$= \{x_2\}$	Lem. E.3 (3) and Assump. in lines 12-14
$loops\ (x_1 \mapsto x_2 \bullet \gamma_1 \backslash x_1) =$	
$= loops\ (x_1 \mapsto x_2) \cup loops\ (\gamma_1 \backslash x_1)$	Distrib. of *loops*
$= loops\ (\gamma_1 \backslash x_1)$	Def. of *loops*
$= \emptyset$	Assump. in lines 12-14
$nodes\ (x_1 \mapsto x_2 \bullet \gamma_1 \backslash x_1) =$	
$= nodes\ (x_1 \mapsto x_2) \cup nodes\ (\gamma_1 \backslash x_1)$	Distrib. of *nodes*
$= \{x\} \cup nodes\ (\gamma_1 \backslash x_1)$	Def. of *nodes*
$= nodes\ (x_1 \mapsto x_1) \cup nodes\ (\gamma_1 \backslash x_1)$	Def. of *nodes*
$= nodes\ (x_1 \mapsto x_1 \bullet \gamma_1 \backslash x_1)$	Distrib. of *nodes*
$= S_1$	Assump. in lines 12-14

The last step in the proof outline starting in line 20 is joining both subgraphs $x_1 \mapsto x_2 \bullet \gamma_1 \backslash x_1$ and γ_2. Both *loops* and *nodes* use plain distributivity. In contrast, for *summits*, instead of proving that the joined graph is preacyclic in order to apply distributivity, we exploit the fact that both

subgraphs have the same *summits* and apply lemma E.3 (2) directly.

$$
\begin{aligned}
&\mathit{loops}\ (x_1 \mapsto x_2 \bullet \gamma_1 \backslash x_1 \bullet \gamma_2) = \\
&\quad = \mathit{loops}\ (x_1 \mapsto x_2 \bullet \gamma_1 \backslash x_1) \cup \mathit{loops}\ \gamma_2 && \text{Distrib. of } \mathit{loops} \\
&\quad = \{x_2\} && \text{Assump. in lines 12-14}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{nodes}\ (x_1 \mapsto x_2 \bullet \gamma_1 \backslash x_1 \bullet \gamma_2) = \\
&\quad = \mathit{nodes}\ (x_1 \mapsto x_2 \bullet \gamma_1 \backslash x_1) \cup \mathit{nodes}\ \gamma_2 && \text{Distrib. of } \mathit{nodes} \\
&\quad = S_1 \cup S_2 && \text{Assump. in lines 12-14}
\end{aligned}
$$

The proof concludes in line 24 by folding the desired spatial predicate.