```
struct Node {
    int m;
    struct Node * l;
    struct Node * r; };

// We use R to represent reachable(γ, x)

void spanning(struct Node * x) {
// {graph(x, γ) ∧ γ(x).1 = 0}
    struct Node * l, * r; int root_mark;
// {graph(x, γ) ∧ ∃l, r. γ(x) = (0, l, r)}
// {graph(x, γ) ∧ γ(x) = (0, l, r)}
// {vertices_at(reachable(γ, x), γ) ∧ γ(x) = (0, l, r)}
// {vertices_at(R, γ) ∧ γ(x) = (0, l, r)}
// ↘ {x ↦ 0, l, r ∧ γ(x) = (0, l, r)}
    l = x -> l;
    r = x -> r;
    x -> m = 1;
// ↙ {x ↦ 1, l, r ∧ γ(x) = (0, l, r) ∧ ∃γ1. mark1(γ, x, γ1)}
// {∃γ1. vertices_at(R, γ1) ∧ γ(x) = (0, l, r) ∧ mark1(γ, x, γ1)}
// {vertices_at(R, γ1) ∧ γ(x) = (0, l, r) ∧ mark1(γ, x, γ1)}
    if (l) {
        root_mark = l -> m;
        if (root_mark == 0) {
            spanning(l);
        } else { x -> l = 0; } }
// { ∃γ2. vertices_at(R, γ2) ∧ γ(x) = (0, l, r) ∧
//   mark1(γ, x, γ1) ∧ e_span(γ1, x.L, γ2) }
// { vertices_at(R, γ2) ∧ γ(x) = (0, l, r) ∧
//   mark1(γ, x, γ1) ∧ e_span(γ1, x.L, γ2) }
    if (r) {
        root_mark = r -> m;
        if (root_mark == 0) {
            spanning(r);
        } else { x -> r = 0; } }
// { ∃γ3. vertices_at(R, γ3) ∧ γ(x) = (0, l, r) ∧
//   mark1(γ, x, γ1) ∧ e_span(γ1, x.L, γ2) ∧ e_span(γ2, x.R, γ3) }
} // {∃γ3. vertex_at(reachable(γ, x), γ3) ∧ span(γ, x, γ3)}
```

$$\mathsf{vertices\_at}(\mathsf{reachable}(\gamma_1, \mathrm{x}), \gamma_2) \triangleq \bigstar_{v \in \mathsf{reachable}(\gamma_1, \mathrm{x})} v \mapsto \gamma_2(v)$$

$$\begin{aligned} span(\gamma_1, \mathrm{x}, \gamma_2) \triangleq\ & mark(\gamma_1, \mathrm{x}, \gamma_2) \wedge \gamma_1 \uparrow (\lambda v. x \overset{\gamma_1}{\rightsquigarrow}_0^\star v) \text{ is a tree} \wedge \\ & \gamma_1 \uparrow (\lambda v. \neg x \overset{\gamma_1}{\rightsquigarrow}_0^\star v) = \gamma_2 \uparrow (\lambda v. \neg x \overset{\gamma_1}{\rightsquigarrow}_0^\star v) \wedge \\ & (\forall v.\ x \overset{\gamma_1}{\rightsquigarrow}_0^\star v \Rightarrow \gamma_2 \models x \rightsquigarrow v) \wedge \\ & (\forall a, b.\ x \overset{\gamma_1}{\rightsquigarrow}_0^\star a \Rightarrow \neg x \overset{\gamma_1}{\rightsquigarrow}_0^\star b \Rightarrow \neg\gamma_2 \models a \rightsquigarrow b) \end{aligned}$$

$$e\_span(\gamma_1, e, \gamma_2) \triangleq \begin{cases} \gamma_1 - e = \gamma_2 & t(\gamma_1, e) = 1 \\ span(\gamma_1, t(\gamma_1, e), \gamma_2) & t(\gamma_1, e) = 0 \end{cases}$$

Figure 10. Clight code and proof sketch for bigraph spanning tree.

A. Spanning and Copying

In Figure 10 we show a simplified proof script for the spanning tree algorithm. Unlike graph marking, the spanning tree algorithm changes the structure of the graph, leading to a more complicated specification, in both the pure part and the spatial part. Observe that the *span* relation is rather long; the e_span handles the case of either calling spanning tree or deleting an edge.

We put the proof sketch of the graph copying algorithm in Figure 11 and Figure 12. Just like other parts of the paper, both algorithms have been machine verified.

```
1  struct Node {
2      int m;
3      struct Node * l;
4      struct Node * r; };
5
6  // We use x <-γ-> x' to represent x = x' = 0 ∨ γ(x) = (x', _, _)
7
8  struct Node * copy(struct Node * x) {
9      struct Node * l, * r, * x0, * l0, * r0;
10 // {graph(x, γ)}
11        if (x == 0)
12          return 0;
13 // {graph(x, γ) ∧ x ≠ 0}
14 // {graph(x, γ) ∧ ∃x0, l, r. γ(x) = (x0, l, r)}
15 // {graph(x, γ) ∧ γ(x) = (x0, l, r)}
16 // ↘ {x ↦ x0, l, r ∧ γ(x) = (x0, l, r)}
17        x0 = x -> m;
18 // ↙ {x ↦ x0, l, r ∧ γ(x) = (x0, l, r) ∧ x0 = x0}
19 // {graph(x, γ) ∧ γ(x) = (x0, l, r)}
20        if (x0 != 0)
21          return x0;
22 // {graph(x, γ) ∧ γ(x) = (0, l, r)}
23        x0 = (struct Node *) mallocN (sizeof (struct Node));
24 // {graph(x, γ) * x0 ↦ _, _, _ ∧ γ(x) = (0, l, r)}
25 // ↘ {x ↦ 0, l, r * x0 ↦ _, _, _ ∧ γ(x) = (0, l, r)}
26        l = x -> l;
27        r = x -> r;
28        x -> m = x0;
29        x0 -> m = 0;
30 // ↙ { x ↦ x0, l, r * x0 ↦ 0, _, _ ∧
       γ(x) = (0, l, r) ∧ ∃γ1 γ1'. v_copy1(γ, x, γ1, γ1') }
31 // { ∃γ1 γ1'. graph(x, γ1) * x0 ↦ 0, _, _ ∧
       γ(x) = (0, l, r) ∧ v_copy1(γ, x, γ1, γ1') }
32 // { graph(x, γ1) * x0 ↦ 0, _, _ ∧
       γ(x) = (0, l, r) ∧ v_copy1(γ, x, γ1, γ1') }
33 // { graph(x, γ1) * x0 ↦ 0, _, _ * holegraph(x0, γ1') ∧
       γ(x) = (0, l, r) ∧ v_copy1(γ, x, γ1, γ1') }
34 // ↘ {graph(l, γ1)}
35        l0 = copy(l);
36 // ↙ { ∃γ2 γ2''. graph(l, γ2) * graph(l0, γ2'') ∧
       copy(γ1, l, γ2, γ2'') ∧ l <-γ2-> l0 }
37 // { ∃γ2 γ2''. graph(x, γ2) * x0 ↦ 0, _, _ * holegraph(x0, γ1') *
       graph(l0, γ2'') ∧ γ(x) = (0, l, r) ∧ v_copy1(γ, x, γ1, γ1') ∧
       copy(γ1, l, γ2, γ2'') ∧ l <-γ2-> l0 }
38 // { ∃γ2 γ2'. graph(x, γ2) * x0 ↦ 0, _, _ * holegraph(x0, γ2') ∧
       γ(x) = (0, l, r) ∧ v_copy1(γ, x, γ1, γ1') ∧
       e_copy(γ1, γ1', x.L, γ2, γ2') ∧ l <-γ2-> l0 }
39 // { graph(x, γ2) * x0 ↦ 0, _, _ * holegraph(x0, γ2') ∧
       γ(x) = (0, l, r) ∧ v_copy1(γ, x, γ1, γ1') ∧
       e_copy(γ1, γ1', x.L, γ2, γ2') ∧ l <-γ2-> l0 }
40        x0 -> l = l0;
41 // { graph(x, γ2) * x0 ↦ 0, l0, _ * holegraph(x0, γ2') ∧
       γ(x) = (0, l, r) ∧ v_copy1(γ, x, γ1, γ1') ∧
       e_copy(γ1, γ1', x.L, γ2, γ2') ∧ l <-γ2-> l0 }
42 // ↘ {graph(r, γ2)}
43        r0 = copy(r);
44 // ↙ { ∃γ3 γ3''. graph(r, γ3) * graph(r0, γ3'') ∧
       copy(γ2, r, γ3, γ3'') ∧ r <-γ3-> r0 }
45 // { ∃γ3 γ3''. graph(x, γ3) * x0 ↦ 0, l0, _ * holegraph(x0, γ3') *
       graph(r0, γ3'') ∧ γ(x) = (0, l, r) ∧ v_copy1(γ, x, γ1, γ1') ∧
       e_copy(γ1, γ1', x.L, γ2, γ2') ∧ copy(γ2, r, γ3, γ3'') ∧
       l <-γ2-> l0 ∧ r <-γ3-> r0 }
46 // { ∃γ3 γ3'. graph(x, γ3) * x0 ↦ 0, l0, _ * holegraph(x0, γ3') ∧
       γ(x) = (0, l, r) ∧ v_copy1(γ, x, γ1, γ1') ∧
       e_copy(γ1, γ1', x.L, γ2, γ2') ∧ e_copy(γ2, γ2', x.R, γ3, γ3') ∧
       l <-γ2-> l0 ∧ r <-γ3-> r0 }
```

Figure 11. Proof sketch for bigraph copy - part 1

```
1  // { graph(x, γ3) * x0 ↦ 0, l0, _ * holegraph(x0, γ3') ∧
       γ(x) = (0, l, r) ∧ v_copy1(γ, x, γ1, γ1') ∧
       e_copy(γ1, γ1', x.L, γ2, γ2') ∧ e_copy(γ2, γ2', x.R, γ3, γ3') ∧
       l <-γ2-> l0 ∧ r <-γ3-> r0 }
2        x0 -> r = r0;
3  // { graph(x, γ3) * x0 ↦ 0, l0, r0 * holegraph(x0, γ3') ∧
       γ(x) = (0, l, r) ∧ v_copy1(γ, x, γ1, γ1') ∧
       e_copy(γ1, γ1', x.L, γ2, γ2') ∧ e_copy(γ2, γ2', x.R, γ3, γ3') ∧
       l <-γ2-> l0 ∧ r <-γ3-> r0 }
4  // { graph(x, γ3) * graph(x0, γ3') ∧ copy(γ, x, γ3, γ3') ∧ x <-γ3-> x0 }
```

$$\mathsf{holegraph}(x,\gamma) \triangleq \bigstar_{v \in \mathsf{reachable}(\gamma,x)-\{x\}} v \mapsto \gamma(v)$$

$$\begin{aligned}
iso(f_V, f_E, \gamma_1, \gamma_2) \triangleq\ & f_V \text{ is a bijection between } \phi_V(\gamma_1) \text{ and } \phi_V(\gamma_2) \wedge \\
& f_E \text{ is a bijection between } \phi_E(\gamma_1) \text{ and } \phi_E(\gamma_2) \wedge \\
& \forall e, f_V(s(\gamma_1, e)) = s(\gamma_2, f_E(e)) \wedge \\
& \forall e, f_V(d(\gamma_1, e)) = d(\gamma_2, f_E(e))
\end{aligned}$$

$$\begin{aligned}
v_copy1(\gamma_1, x, \gamma_2, \gamma_2') \triangleq\ & \exists x'. x \neq 0 \wedge mark1(\gamma_1, x, \gamma_2) \wedge \\
& x \xleftrightarrow{\gamma_2} x' \wedge \gamma_2' = \{x_0\}
\end{aligned}$$

$$\begin{aligned}
copy(\gamma_1, x, \gamma_2, \gamma_2') \triangleq\ & mark(\gamma_1, x, \gamma_2) \wedge \\
& \exists f_V\, f_E. iso(f_V, f_E, \gamma_2 \uparrow (\lambda v. x \overset{\gamma_2}{\rightsquigarrow}_0^\star v), \gamma_2') \wedge \\
& \forall x x'. f_V(x) = x' \Leftrightarrow x \xleftrightarrow{\gamma_2} x'
\end{aligned}$$

$$\begin{aligned}
e_copy(\gamma_1, \gamma_1', e, \gamma_2, \gamma_2') \triangleq\ & \exists \gamma_2''. \gamma_2' = \gamma_1' + \gamma_2'' \wedge \\
& mark(\gamma_1, x, \gamma_2) \wedge \exists f_V\, f_E. \\
& iso(f_V, f_E, \{e\} + \gamma_2 \uparrow (\lambda v. x \overset{\gamma_2}{\rightsquigarrow}_0^\star v), \gamma_2'') \wedge \\
& \forall x x'. f_V(x) = x' \Leftrightarrow x \xleftrightarrow{\gamma_2} x'
\end{aligned}$$

Here, when we mention *mark* and *mark1*, the value 1s in the original definition are changed to non-zero values.

Figure 12. Proof sketch for bigraph copy - part 2

B. Proof of RAMIFY-P and RAMIFY-PQ

See Figure 13 for the proofs of RAMIFY-P and RAMIFY-PQ.

C. Difficulty using graph_T

See Figure 14 for an attempt to prove the entailment $100 \mapsto 42, 100, 0 \vdash \mathsf{graph}_T(100, \hat{\gamma})$. Part of the problem is that the recursive structure interacts very badly with $\circledast$: if the recursion involved $*$ then it **would** be provable, by induction on the finite memory (each "recursive call" would be on a strictly smaller subheap). This is why Knaster-Tarski works so well with list, tree, and DAG predicates in separation logic.

Proof of RAMIFY-P from FRAME and CONSEQUENCE:

$$\dfrac{G_1 \vdash L_1 * [\![c]\!](L_2 \mathrel{-\!\!*} G_2) \qquad \dfrac{\{L_1\}\ c\ \{L_2\}}{\{L_1 * [\![c]\!](L_2 \mathrel{-\!\!*} G_2)\}\ c\ \{L_2 * [\![c]\!](L_2 \mathrel{-\!\!*} G_2)\}}\ (1) \qquad \dfrac{\dfrac{\overset{\langle c \rangle}{\cong} \text{ is reflexive}}{[\![c]\!](L_2 \mathrel{-\!\!*} G_2) \vdash L_2 \mathrel{-\!\!*} G_2}\ (2)}{L_2 * [\![c]\!](L_2 \mathrel{-\!\!*} G_2) \vdash G_2}\ (3)}{\{G_1\}\ c\ \{G_2\}}$$

(1) $\forall P.\ [\![c]\!]P$ ignores $\mathsf{FreeVar}(c)$ (2) axiom T of modal logic (3) $(P * Q \vdash R) \Leftrightarrow (P \vdash Q \mathrel{-\!\!*} R)$

Proof of RAMIFY-PQ from RAMIFY-P:

$$\dfrac{\{L_1\}\ c\ \{\exists x.\ L_2\} \qquad \dfrac{G_1 \vdash L_1 * [\![c]\!]\big(\forall x.\ (L_2 \mathrel{-\!\!*} G_2)\big) \qquad \dfrac{\dfrac{\vdots}{\forall x.\ (L_2 \mathrel{-\!\!*} G_2) \vdash (\exists x.\ L_2) \mathrel{-\!\!*} (\exists x.\ G_2)}\ (1)}{[\![c]\!]\big(\forall x.\ (L_2 \mathrel{-\!\!*} G_2)\big) \vdash [\![c]\!]\big((\exists x.\ L_2) \mathrel{-\!\!*} (\exists x.\ G_2)\big)}\ (2)}{L_1 * [\![c]\!]\big(\forall x.\ (L_2 \mathrel{-\!\!*} G_2)\big) \vdash L_1 * [\![c]\!]\big((\exists x.\ L_2) \mathrel{-\!\!*} (\exists x.\ G_2)\big)}}{G_1 \vdash L_1 * [\![c]\!]\big((\exists x.\ L_2) \mathrel{-\!\!*} (\exists x.\ G_2)\big)}}{\{G_1\}\ c\ \{\exists x.\ G_2\}}$$

(1) tautology using $(P * Q \vdash R) \Leftrightarrow (P \vdash Q \mathrel{-\!\!*} R)$ (2) reduction using modal axioms K and N

Figure 13. Proofs of RAMIFY-P and RAMIFY-PQ

$$\dfrac{\dfrac{100 \mapsto 42, 100, 0 \ \vdash\ 100 \mapsto 42, 100, 0 \mathbin{\text{⍟}} \mathsf{graph}_T(100, \hat{\gamma})}{100 \mapsto 42, 100, 0 \ \vdash\ \hat{\gamma}(100) = (42, 100, 0) \ \wedge\ 100 \mapsto 42, 100, 0 \mathbin{\text{⍟}} \mathsf{graph}_T(100, \hat{\gamma}) \mathbin{\text{⍟}} \mathsf{graph}_T(0, \hat{\gamma})}\ (2)}{100 \mapsto 42, 100, 0 \ \vdash\ \mathsf{graph}_T(100, \hat{\gamma})}\ (1)$$

(1) Unfold graph_T, dismiss first disjunct (contradiction), introduce existentials (which must be 42,100,0)
(2) simplify using $P * \mathsf{emp} \dashv\vdash P$ and remove pure conjunct

Figure 14. An attempt to prove a "simple" entailment

D. Problem with Appel and McAllester's fixpoint

Appel and McAllester proposed another fixpoint μ_{A} that is sometimes used to define recursive predicates in separation logic [1]. This time the functional F_P needs to be *contractive*, which to a first order of approximation means that all recursion needs to be guarded by the "approximation modality" $\triangleright$ [2], *i.e.* our graph predicate would look like

$$\begin{aligned}&\mathsf{graph}_A(x, \gamma) \ \triangleq \\ &(x = 0 \wedge \mathsf{emp}) \vee \exists m, l, r.\ \gamma(x) = (m, l, r) \wedge \\ &\quad x \mapsto m, l, r \mathbin{\text{⍟}} \triangleright \mathsf{graph}_A(l, \gamma) \mathbin{\text{⍟}} \triangleright \mathsf{graph}_A(r, \gamma)\end{aligned}$$

Unfortunately, $\triangleright P$ is not precise for all P, so graph_A is not precise either. The approximation modality's universal imprecision has never been noticed before.