Verifying overlapping datastructures

John Wickerson

Figure 1 shows a simple overlapping datastructure. It is overlapping in the sense that we have two edge sets imposed on the same set of nodes. Each node is a struct comprising a `val`, a brown '`next`' pointer, and a green '`fd`' pointer. The brown `next` pointers form a chain, starting at `start`, of all the nodes in which `val` is non-decreasing. The `mod` array identifies four equivalence classes, obtained by identifying `val` fields modulo 4. The green `fd` pointers form a circular singly-linked list that encompass all of the nodes in the same equivalence class, in any order. We use casting tricks to treat the `start` pointer as a sentinel node in the linked list, and, likewise, each element of `mod` as a sentinel node in its equivalence class.

We now consider several attempts to reason about this datastructure.

1 Inductive predicate for the whole datastructure

We might first try to find a predicate p to describe the structure as a whole, in the same vein as the *list* or *tree* predicates in separation logic. Note that a naïve attempt of the form

$$p(x) \Leftrightarrow \ldots * x.\texttt{next} \mapsto y * x.\texttt{fd} \mapsto z * p(y) * p(z)$$

fails because, unlike for lists or trees, y and z need not be distinct. So the predicate might look more like this:

$$\begin{aligned} p_{m,n}(x) \Leftrightarrow\ & (x.\texttt{val} \mapsto v \wedge v \geq m \wedge v \bmod 4 = n) \\ & * x.\texttt{next} \mapsto y * x.\texttt{fd} \mapsto z * \begin{cases} p_{v,n}(y) & \text{if } y = z \\ p_{v,?}(y) * p_{?,n}(z) & \text{if } y \neq z \end{cases} \end{aligned}$$

But what to put for the two ?s is not clear at all. If node x is in the i^{th} equivalence class, and its `next` pointer points to node y, we do not know anything about y's class – it could be i or any other. Yet not to specify y's class at all would be to make p too weak to capture the intended specification of the datastructure. The problem can be stated by saying that the datastructure is not built inductively.

2 Fractional permissions

Perhaps we are decomposing the structure in the wrong way. Perhaps we should not break the structure down into its individual nodes just yet: instead, let us first separate the two

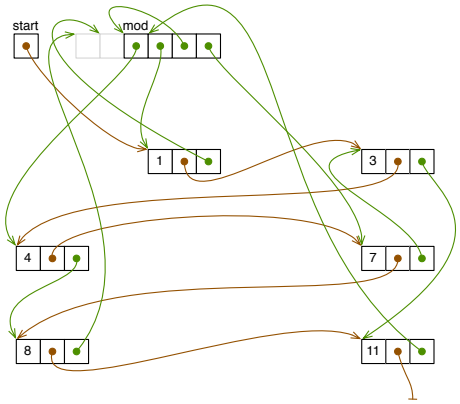

Figure 1: An overlapping datastructure. We have no ownership of grey-outlined cells.

overlapping structures. I'm imagining an overall description of the state of this form:

$$\begin{aligned} \mathit{state} \stackrel{\text{def}}{=} \ & \texttt{start} \mapsto x * \texttt{mod} \mapsto c_0, c_1, c_2, c_3 \\ & * \mathit{list}(x) * \mathit{class}_0(c_0) * \mathit{class}_1(c_1) * \mathit{class}_2(c_2) * \mathit{class}_3(c_3) \end{aligned}$$

How might we split nodes? The `next` field clearly belongs to the *list*, and the `fd` field belongs to the *class*. But the `val` field needs to be available to both: the *list* predicate needs to specify that `val` doesn't decrease, while the *class* predicate needs to know `val` module 4. Let us use fractional permissions: let each predicate reason about half the `val` field. The *list* and *class* predicates can then be implemented as follows:

$$\begin{aligned} \mathit{list}_n(x) &\Leftrightarrow x \doteq 0 \vee \exists v, x'.\, x \dot{\neq} 0 * (x.\texttt{val} \overset{.5}{\mapsto} v \wedge v \geq n) * x.\texttt{next} \mapsto x' * \mathit{list}_v(x') \\ \mathit{list}(x) &\stackrel{\text{def}}{=} \mathit{list}_0(x) \\ \mathit{class}_i(c) &\Leftrightarrow c \doteq \texttt{mod}+i-2 \vee \exists v, c'.\, c \dot{\neq} \texttt{mod}+i-2 * (c.\texttt{val} \overset{.5}{\mapsto} v \wedge v \bmod 4 = i) * c.\texttt{fd} \mapsto c' * \mathit{class}_i(c') \end{aligned}$$

The two overlapping structures are visualised in Fig. 2. It is clear that when the two structures are superimposed, we attain the full picture of Fig. 1.

Unfortunately, there is not enough consistency between the two descriptions. The *state* predicate to specify that the four equivalence classes should contain all of the nodes that are in the increasing list, and that the increasing list should contain all of the nodes that are in one of the four classes. This means that it describes undesirable states such as that depicted in Fig. 3.

3 Enforcing consistency using boxed assertions

To enforce the consistency, I propose using the 'boxed assertions' concept from RGSep. Rather than splitting the `val` field into two half-permissions, let us form a shared region

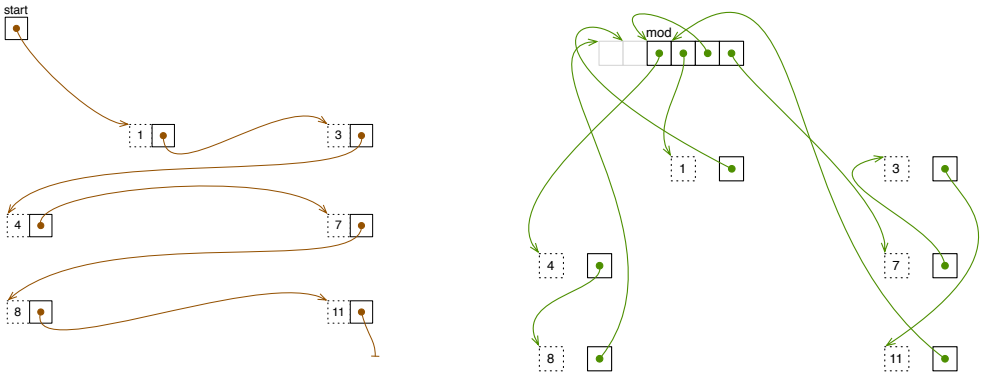

Figure 2: The *list* and $class_i$ parts of the datastructure. We have half ownership of dotted-outlined cells.

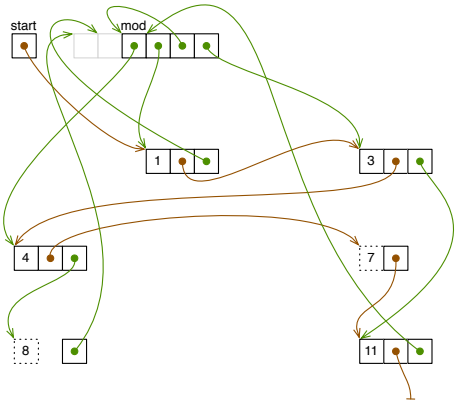

Figure 3: An undesirable state

that comprises all of the `val` fields. If the *list* and *class* predicates form two non-unifiable views of the `val` fields, then when they come to be composed, a contradiction will ensue. For instance, if x_n is the address of the node with value n (assuming that all values are distinct) then, in our undesirable state of Fig. 3, the *list* predicate would describe the shared region as

$$\boxed{x_1.\texttt{val} \mapsto 1 * x_3.\texttt{val} \mapsto 3 * x_4.\texttt{val} \mapsto 4 * x_7.\texttt{val} \mapsto 7 * x_{11}.\texttt{val} \mapsto 11}$$

while the family of $class_i$ predicates would see it as

$$\boxed{x_4.\texttt{val} \mapsto 4 * x_8.\texttt{val} \mapsto 8 * x_1.\texttt{val} \mapsto 1 * x_3.\texttt{val} \mapsto 3 * x_{11}.\texttt{val} \mapsto 11}.$$

The $*$-ing together of these two assertions, say $\boxed{P_{list}}$ and $\boxed{P_{class}}$, is equivalent to $\boxed{P_{list} \wedge P_{class}}$, which is simply $\boxed{\mathsf{false}}$.

Let us introduce a new '$\circledast$' operator to the assertion language. Its main property is that $\boxed{P} \circledast \boxed{Q}$ is equivalent to $\boxed{P * Q}$. Thus, its semantics shall be defined as follows:

$$l, s, i \models p_0 \circledast p_1 \stackrel{\text{def}}{=} \exists l_0, l_1, s_0, s_1.\, l{=}l_0 \odot l_1 \wedge s{=}s_0 \odot s_1 \wedge l_0, s_0, i \models p_0 \wedge l_1, s_1, i \models p_1$$

We can now define the state as follows:

$$\begin{aligned} state \stackrel{\text{def}}{=}\ & \texttt{start} \mapsto x * \texttt{mod} \mapsto c_0, c_1, c_2, c_3 \\ & * list(x) * (class_0(c_0) \circledast class_1(c_1) \circledast class_2(c_2) \circledast class_3(c_3)) \end{aligned}$$

The double-stars on the second line build up the shared state of the family of *class* predicates. The single star between that and the *list* predicate enforces a consistent view of the same shared state. The *list* and *class* predicates can be defined like so.

$$\begin{array}{rcl} list_n(x) & \Leftrightarrow & x \doteq 0 \vee \exists x'.\, x \dot{\neq} 0 * x.\texttt{next} \mapsto x' * (\boxed{x.\texttt{val} \mapsto v \wedge v \geq n} \circledast list_v(x')) \\ list(x) & \stackrel{\text{def}}{=} & list_0(x) \\ class_n(c) & \Leftrightarrow & c \doteq 0 \vee \exists c'.\, c \dot{\neq} 0 * c.\texttt{fd} \mapsto c' * (\boxed{c.\texttt{val} \mapsto v \wedge v \bmod 4 = n} \circledast list_n(c', c)) \\ list_n(c, c') & \Leftrightarrow & c \doteq c' \vee \exists c''.\, c \dot{\neq} c' * c.\texttt{fd} \mapsto c'' * (\boxed{c.\texttt{val} \mapsto v \wedge v \bmod 4 = n} \circledast list_n(c'', c')) \end{array}$$

However, the consistency is only being enforced at the topmost level. Thus, I don't think we get the required compositionality that we would like in order to be able to reason locally. For instance, how might we reason about the following program that updates a node in the structure?

4 A program

The program changes the value of node `c` to `w`, updating its position in the datastructure accordingly.

```
void setNode(node* c, int w) {

  // set c's new value, caching the old one
  int v = c->val;
  c->val = w;

  // remove c from list
  node* p = (node*) (&start - 1);
  while (p->next != c) p = p->next;
  p->next = c->next;

  // return c to list
  p = (node*) (&start - 1);
  while ((p->next != 0) && (p->next)->val < w) p = p->next;
  c->next = p->next;
  p->next = c;

  // remove c from its old class
  p = (node*) (mod + (v%4) - 2);
  while (p->fd != c) p = p->fd;
  p->fd = c->fd;

  // add c to its new class
  p = (node*) (mod + (w%4) - 2);
  c->fd = p->fd;
  p->fd = c;
}
```

5 Mutually-referring boxed regions

Another approach that might work is to use several different boxes. For this example, we shall have 5 boxes: Boxes 0-3 for each of the classes, and Box 4 for the linked list. Box 4 shall describe a state such as that on the left of Fig. 2: the `next` pointers plus half-permission on the `val` fields. Boxes 0-3 shall each describe one of the four separate sections on the right of Fig. 2: the `fd` pointers plus the other half-permission on the `val` fields.

Every part of the state thus lies within one of these shared regions: indeed, since they are mutually disjoint, every part of the state lies in *exactly* one of these regions. So we can describe the entire state by $*$-conjoining the five predicates together:

$$state \stackrel{\text{def}}{=} \boxed{class_0}_0 * \boxed{class_1}_1 * \boxed{class_2}_2 * \boxed{class_3}_3 * \boxed{list}_4$$

We define the $list$ and $class_i$ predicates like so:

$$
\begin{array}{rcl}
\mathit{lel}(x,y) & \Leftrightarrow & \exists m,n.\, x.\texttt{val} \overset{.25}{\mapsto} m \\
& & * \; x.\texttt{next} \mapsto y * (y \dot{=} 0 \vee y \dot{\neq} 0 * y.\texttt{val} \overset{.25}{\mapsto} n * m \dot{\leq} n) \\
\mathit{Lel}(x,y) & \Leftrightarrow & \exists m,n,i.\, x.\texttt{val} \overset{.25}{\mapsto} m \\
& & * \; x.\texttt{next} \mapsto y * (y \dot{=} 0 \vee y \dot{\neq} 0 * y.\texttt{val} \overset{.25}{\mapsto} n * m \dot{\leq} n) \\
& & * \; i \dot{=} m\%4 * \boxed{\mathit{inclass}_i(x)}_i \\
\mathit{cel}_i(x,y) & \Leftrightarrow & \exists n.\, x.\texttt{val} \overset{.5}{\mapsto} n * n\%4 \dot{=} i * x.\texttt{fd} \mapsto y \\
\mathit{Cel}_i(x,y) & \Leftrightarrow & \exists n.\, x.\texttt{val} \overset{.5}{\mapsto} n * n\%4 \dot{=} i * x.\texttt{fd} \mapsto y \\
& & * \; \boxed{\mathit{inlist}(x)}_4 \\
\mathit{inlist}(y) & \Leftrightarrow & \exists x,z.\, \texttt{\&start} \mapsto x * x.\texttt{val} \overset{.25}{\mapsto} \_ * \mathit{lel}^*(x,y) * \mathit{lel}^+(y,0) \\
\mathit{inclass}_i(x) & \Leftrightarrow & \exists y.\, \texttt{mod+}i \mapsto y * \mathit{cel}_i^*(y,x) * \mathit{cel}_i^+(x, \texttt{mod+}i{-}2) \\
\mathit{list} & \Leftrightarrow & \exists y.\, \texttt{\&start} \mapsto y * (y \dot{=} 0 \vee y \dot{\neq} 0 * y.\texttt{val} \overset{.25}{\mapsto} \_) * \mathit{Lel}^*(y,0) \\
\mathit{class}_i & \Leftrightarrow & \exists y.\, \texttt{mod+}i \mapsto y * \mathit{Cel}_i^*(y, \texttt{mod+}i{-}2)
\end{array}
$$

Note the following points:

- 'Lel' abbreviates 'list element' and 'cel' abbreviates 'class element'.
- If R is a binary predicate, then R^* denotes its reflexive transitive closure, defined as the least relation satisfying:

  $$R^*(x,y) \;\Leftrightarrow\; x \dot{=} y \vee \exists x'.\, x \dot{\neq} y * R(x,x') * R^*(x',y)$$

  Then $R^+(x,y)$ means $\exists x'.\, x \dot{\neq} y * R(x,x') * R^*(x',y)$.
- Those predicates that are capitalised carry cross references to other boxed regions. For instance, $\mathit{Lel}(x,y)$ is the same as $\mathit{lel}(x,y)$, except that it additionally states that x appears in its respective equivalence class. Note that $\mathit{inclass}_i(x)$ uses the non-capitalised predicates, so as to avoid circular cross-references.
- We use quarter-permission pointers in the *lel* and *Lel* predicates. The *list* predicate has a total of half-permission on the `val` fields. The *lel* and *Lel* predicates further split this permission in half, because they need to access each `val` field twice: once to compare it to the previous node's `val`, and once to compare it to the next node's `val`. As a result of this, we need to make up the missing quarter-permission on the very first `val`, and this is done in the definitions of *list* and *inlist*.

6 Verifying the program

Lemma: $R^*(x,y) \;\Leftrightarrow\; x \dot{=} y \vee \exists y'.\, x \dot{\neq} y * R^*(x,y') * R(y',y)$

$\{\boxed{\mathit{class}_0}_0 * \boxed{\mathit{class}_1}_1 * \boxed{\mathit{class}_2}_2 * \boxed{\mathit{class}_3}_3 * \boxed{\mathit{list}}_4 * \exists i \in [0,4).\, \boxed{\mathit{inclass}_i(\texttt{c})}_i * \boxed{\mathit{inlist}(\texttt{c})}_4\}$

```
// Note the slight weakness in the statement: c must
// be in some class, but not necessarily the right
// one. (Actually, it must be in the right one, because
```

```
// we have additionally the class0, class1, class2 and
// class3 predicates that enforce the well-formedness
// of the classes. Perhaps the above definitions should
// be 'weakened' in a similar way?)
void setNode(node* c, int w) {
```

$$\left\{ \begin{array}{l} \boxed{class_0}_0 * \boxed{class_1}_1 * \boxed{class_2}_2 * \boxed{class_3}_3 * \boxed{list}_4 * \exists i \in [0,4).\, \boxed{inclass_i(\mathtt{c})}_i \\ * \boxed{\exists s, n.\, \mathtt{\&start} \mapsto s * s.\mathtt{val} \overset{.25}{\mapsto} \_ * lel^*(s, \mathtt{c}) * lel(\mathtt{c}, n) * lel^*(n, 0)}_4 \end{array} \right\}$$

$$\left\{ \begin{array}{l} \boxed{class_0}_0 * \boxed{class_1}_1 * \boxed{class_2}_2 * \boxed{class_3}_3 * \boxed{list}_4 * \exists i \in [0,4).\, \boxed{inclass_i(\mathtt{c})}_i \\ * \boxed{\begin{array}{l} \exists s, n, v, v'.\, \mathtt{\&start} \mapsto s * s.\mathtt{val} \overset{.25}{\mapsto} \_ * lel^*(s, \mathtt{c}) * \mathtt{c.val} \overset{.25}{\mapsto} v \\ * \mathtt{c.next} \mapsto n * (n \doteq 0 \vee n \neq 0 * n.\mathtt{val} \overset{.25}{\mapsto} v' * v \leq v') * lel^*(n, 0) \end{array}}_4 \end{array} \right\}$$

```
  // Use Lemma from above.
```

$$\left\{ \begin{array}{l} \boxed{class_0}_0 * \boxed{class_1}_1 * \boxed{class_2}_2 * \boxed{class_3}_3 * \boxed{list}_4 * \exists i \in [0,4).\, \boxed{inclass_i(\mathtt{c})}_i \\ * \boxed{\begin{array}{l} \exists s, p, n, v, v'.\, \mathtt{\&start} \mapsto s * s.\mathtt{val} \overset{.25}{\mapsto} \_ \\ * (s \doteq \mathtt{c} \vee s \neq \mathtt{c} * lel^*(s, p) * lel(p, \mathtt{c})) \\ * \mathtt{c.val} \overset{.25}{\mapsto} v * \mathtt{c.next} \mapsto n \\ * (n \doteq 0 \vee n \neq 0 * n.\mathtt{val} \overset{.25}{\mapsto} v' * v \leq v') * lel^*(n, 0) \end{array}}_4 \end{array} \right\}$$

$$\left\{ \begin{array}{l} \boxed{class_0}_0 * \boxed{class_1}_1 * \boxed{class_2}_2 * \boxed{class_3}_3 * \boxed{list}_4 * \exists i \in [0,4).\, \boxed{inclass_i(\mathtt{c})}_i \\ * \left(\boxed{\begin{array}{l} \exists n, v, v'.\, \mathtt{\&start} \mapsto \mathtt{c} * \mathtt{c.val} \overset{.5}{\mapsto} v * \mathtt{c.next} \mapsto n \\ * (n \doteq 0 \vee n \neq 0 * n.\mathtt{val} \overset{.25}{\mapsto} v' * v \leq v') * lel^*(n, 0) \end{array}}_4 \right. \vee \boxed{\begin{array}{l} \exists s, p, n, v, v'.\, \mathtt{\&start} \mapsto s * s.\mathtt{val} \overset{.25}{\mapsto} \_ \\ * s \neq \mathtt{c} * lel^*(s, p) * lel(p, \mathtt{c}) \\ * \mathtt{c.val} \overset{.25}{\mapsto} v * \mathtt{c.next} \mapsto n \\ * (n \doteq 0 \vee n \neq 0 * n.\mathtt{val} \overset{.25}{\mapsto} v' * v \leq v') * lel^*(n, \end{array}} \end{array} \right\}$$

$$\left\{ \begin{array}{l} \boxed{class_0}_0 * \boxed{class_1}_1 * \boxed{class_2}_2 * \boxed{class_3}_3 * \boxed{list}_4 * \exists i \in [0,4).\, \boxed{inclass_i(\mathtt{c})}_i \\ * \left(\boxed{\begin{array}{l} \exists n, v, v'.\, \mathtt{\&start} \mapsto \mathtt{c} * \mathtt{c.val} \overset{.5}{\mapsto} v * \mathtt{c.next} \mapsto n \\ * (n \doteq 0 \vee n \neq 0 * n.\mathtt{val} \overset{.25}{\mapsto} v' * v \leq v') * lel^*(n, 0) \end{array}}_4 \right. \vee \boxed{\begin{array}{l} \exists s, p, n, u, v, v'', v'.\, \mathtt{\&start} \mapsto s * s.\mathtt{val} \overset{.25}{\mapsto} \_ \\ * s \neq \mathtt{c} * lel^*(s, p) * p.\mathtt{val} \overset{.25}{\mapsto} u * p.\mathtt{next} \mapsto \mathtt{c} \\ * (\mathtt{c} \doteq 0 \vee \mathtt{c} \neq 0 * \mathtt{c.val} \overset{.25}{\mapsto} v'' * u \leq v'') \\ * \mathtt{c.val} \overset{.25}{\mapsto} v * \mathtt{c.next} \mapsto n \\ * (n \doteq 0 \vee n \neq 0 * n.\mathtt{val} \overset{.25}{\mapsto} v' * v \leq v') * lel^*(n, \end{array}} \end{array} \right\}$$

$$\left\{ \begin{array}{l} \boxed{class_0}_0 * \boxed{class_1}_1 * \boxed{class_2}_2 * \boxed{class_3}_3 * \boxed{list}_4 * \exists i \in [0,4).\, \boxed{inclass_i(\mathtt{c})}_i \\ * \left(\boxed{\begin{array}{l} \exists n, v, v'.\, \mathtt{\&start} \mapsto \mathtt{c} * \mathtt{c.val} \overset{.5}{\mapsto} v * \mathtt{c.next} \mapsto n \\ * (n \doteq 0 \vee n \neq 0 * n.\mathtt{val} \overset{.25}{\mapsto} v' * v \leq v') * lel^*(n, 0) \end{array}}_4 \right. \vee \boxed{\begin{array}{l} \exists s, p, n, u, v, v'.\, \mathtt{\&start} \mapsto s * s.\mathtt{val} \overset{.25}{\mapsto} \_ \\ * s \neq \mathtt{c} * lel^*(s, p) * p.\mathtt{val} \overset{.25}{\mapsto} u * p.\mathtt{next} \mapsto \mathtt{c} \\ * \mathtt{c.val} \overset{.5}{\mapsto} v * u \leq v * \mathtt{c.next} \mapsto n \\ * (n \doteq 0 \vee n \neq 0 * n.\mathtt{val} \overset{.25}{\mapsto} v' * v \leq v') * lel^*(n, \end{array}} \end{array} \right\}$$

$$\left\{ \begin{array}{l} \boxed{class_0}_0 * \boxed{class_1}_1 * \boxed{class_2}_2 * \boxed{class_3}_3 * \boxed{list}_4 * \exists i \in [0,4).\, \boxed{inclass_i(\mathtt{c})}_i \\ * \left(\boxed{\begin{array}{l} \exists p, n, u, v, v'.\, (\mathtt{\&start} \mapsto \mathtt{c} \\ \vee\, \mathtt{\&start} \mapsto s * s \neq \mathtt{c} * s.\mathtt{val} \overset{.25}{\mapsto} \_ * lel^*(s, p) * p.\mathtt{val} \overset{.25}{\mapsto} u * u \leq v * p.\mathtt{next} \mapsto \mathtt{c}) \\ * \mathtt{c.val} \overset{.5}{\mapsto} v * \mathtt{c.next} \mapsto n * (n \doteq 0 \vee n \neq 0 * n.\mathtt{val} \overset{.25}{\mapsto} v' * v \leq v') * lel^*(n, 0) \end{array}}_4 \right) \end{array} \right\}$$

```
  // We've now accumulated half permission on c.val in both disjuncts.
  // Now we need the other half from one of the classes.
```

$$\left\{\begin{array}{l}\exists v, i.\, i \dot{=} v\%4 * \boxed{class_0}_0 * \boxed{class_1}_1 * \boxed{class_2}_2 * \boxed{class_3}_3 * \boxed{list}_4 \\ * \left(\begin{array}{l}\exists p, n, u, v'.\, (\texttt{\&start} \mapsto \texttt{c} \\ \quad \vee \texttt{\&start} \mapsto s * s \dot{\neq} \texttt{c} * s.\texttt{val} \overset{.25}{\mapsto} \_ * lel^*(s,p) * p.\texttt{val} \overset{.25}{\mapsto} u * u \dot{\leq} v * p.\texttt{next} \mapsto \texttt{c}) \\ \quad * \texttt{c.val} \overset{.5}{\mapsto} v * \texttt{c.next} \mapsto n * (n \dot{=} 0 \vee n \dot{\neq} 0 * n.\texttt{val} \overset{.25}{\mapsto} v' * v \dot{\leq} v') * lel^*(n,0)\end{array}\right)_4 \\ * \boxed{inclass_i(\texttt{c})}_i\end{array}\right\}$$

$$\left\{\begin{array}{l}\exists v, i.\, i \dot{=} v\%4 * \boxed{class_0}_0 * \boxed{class_1}_1 * \boxed{class_2}_2 * \boxed{class_3}_3 * \boxed{list}_4 \\ * \left(\begin{array}{l}\exists p, n, u, v'.\, (\texttt{\&start} \mapsto \texttt{c} \\ \quad \vee \texttt{\&start} \mapsto s * s \dot{\neq} \texttt{c} * s.\texttt{val} \overset{.25}{\mapsto} \_ * lel^*(s,p) * p.\texttt{val} \overset{.25}{\mapsto} u * u \dot{\leq} v * p.\texttt{next} \mapsto \texttt{c}) \\ \quad * \texttt{c.val} \overset{.5}{\mapsto} v * \texttt{c.next} \mapsto n * (n \dot{=} 0 \vee n \dot{\neq} 0 * n.\texttt{val} \overset{.25}{\mapsto} v' * v \dot{\leq} v') * lel^*(n,0)\end{array}\right)_4 \\ * \boxed{\exists m.\, \texttt{mod}{+}i \mapsto m * cel_i^*(m, \texttt{c}) * cel_i^+(\texttt{c}, \texttt{mod}{+}i{-}2)}_i\end{array}\right\}$$

$$\left\{\begin{array}{l}\exists v, i.\, i \dot{=} v\%4 * \boxed{class_0}_0 * \boxed{class_1}_1 * \boxed{class_2}_2 * \boxed{class_3}_3 * \boxed{list}_4 \\ * \left(\begin{array}{l}\exists p, n, u, v'.\, (\texttt{\&start} \mapsto \texttt{c} \\ \quad \vee \texttt{\&start} \mapsto s * s \dot{\neq} \texttt{c} * s.\texttt{val} \overset{.25}{\mapsto} \_ * lel^*(s,p) * p.\texttt{val} \overset{.25}{\mapsto} u * u \dot{\leq} v * p.\texttt{next} \mapsto \texttt{c}) \\ \quad * \texttt{c.val} \overset{.5}{\mapsto} v * \texttt{c.next} \mapsto n * (n \dot{=} 0 \vee n \dot{\neq} 0 * n.\texttt{val} \overset{.25}{\mapsto} v' * v \dot{\leq} v') * lel^*(n,0)\end{array}\right)_4 \\ * \boxed{\exists m, f.\, \texttt{mod}{+}i \mapsto m * cel_i^*(m, \texttt{c}) * cel_i(\texttt{c}, f) * cel_i^*(f, \texttt{mod}{+}i{-}2)}_i\end{array}\right\}$$

$$\left\{\begin{array}{l}\exists v, i.\, i \dot{=} v\%4 * \boxed{class_0}_0 * \boxed{class_1}_1 * \boxed{class_2}_2 * \boxed{class_3}_3 * \boxed{list}_4 \\ * \left(\begin{array}{l}\exists p, n, u, v'.\, (\texttt{\&start} \mapsto \texttt{c} \\ \quad \vee \texttt{\&start} \mapsto s * s \dot{\neq} \texttt{c} * s.\texttt{val} \overset{.25}{\mapsto} \_ * lel^*(s,p) * p.\texttt{val} \overset{.25}{\mapsto} u * u \dot{\leq} v * p.\texttt{next} \mapsto \texttt{c}) \\ \quad * \texttt{c.val} \overset{.5}{\mapsto} v * \texttt{c.next} \mapsto n * (n \dot{=} 0 \vee n \dot{\neq} 0 * n.\texttt{val} \overset{.25}{\mapsto} v' * v \dot{\leq} v') * lel^*(n,0)\end{array}\right)_4 \\ * \boxed{\exists m, f.\, \texttt{mod}{+}i \mapsto m * cel_i^*(m, \texttt{c}) * \texttt{c.val} \overset{.5}{\mapsto} v * \texttt{c.fd} \mapsto f * cel_i^*(f, \texttt{mod}{+}i{-}2)}_i\end{array}\right\}$$

```
// We now have complete ownership of c.val (although
// it is spread across two different boxed regions

// set c's new value, caching the old one
int v = c->val;
```

$$\left\{\begin{array}{l}\exists i.\, i \dot{=} \texttt{v}\%4 * \boxed{class_0}_0 * \boxed{class_1}_1 * \boxed{class_2}_2 * \boxed{class_3}_3 * \boxed{list}_4 \\ * \left(\begin{array}{l}\exists p, n, u, v'.\, (\texttt{\&start} \mapsto \texttt{c} \\ \quad \vee \texttt{\&start} \mapsto s * s \dot{\neq} \texttt{c} * s.\texttt{val} \overset{.25}{\mapsto} \_ * lel^*(s,p) * p.\texttt{val} \overset{.25}{\mapsto} u * u \dot{\leq} \texttt{v} * p.\texttt{next} \mapsto \texttt{c}) \\ \quad * \texttt{c.val} \overset{.5}{\mapsto} \texttt{v} * \texttt{c.next} \mapsto n * (n \dot{=} 0 \vee n \dot{\neq} 0 * n.\texttt{val} \overset{.25}{\mapsto} v' * \texttt{v} \dot{\leq} v') * lel^*(n,0)\end{array}\right)_4 \\ * \boxed{\exists m, f.\, \texttt{mod}{+}i \mapsto m * cel_i^*(m, \texttt{c}) * \texttt{c.val} \overset{.5}{\mapsto} \texttt{v} * \texttt{c.fd} \mapsto f * cel_i^*(f, \texttt{mod}{+}i{-}2)}_i\end{array}\right\}$$

```
c->val = w;
```

$$\left\{\begin{array}{l}\exists i.\, i \dot{=} \texttt{v}\%4 * \boxed{class_0}_0 * \boxed{class_1}_1 * \boxed{class_2}_2 * \boxed{class_3}_3 * \boxed{list}_4 \\ * \left(\begin{array}{l}\exists p, n, u, v'.\, (\texttt{\&start} \mapsto \texttt{c} \\ \quad \vee \texttt{\&start} \mapsto s * s \dot{\neq} \texttt{c} * s.\texttt{val} \overset{.25}{\mapsto} \_ * lel^*(s,p) * p.\texttt{val} \overset{.25}{\mapsto} u * u \dot{\leq} \texttt{v} * p.\texttt{next} \mapsto \texttt{c}) \\ \quad * \texttt{c.val} \overset{.5}{\mapsto} \texttt{w} * \texttt{c.next} \mapsto n * (n \dot{=} 0 \vee n \dot{\neq} 0 * n.\texttt{val} \overset{.25}{\mapsto} v' * \texttt{v} \dot{\leq} v') * lel^*(n,0)\end{array}\right)_4 \\ * \boxed{\exists m, f.\, \texttt{mod}{+}i \mapsto m * cel_i^*(m, \texttt{c}) * \texttt{c.val} \overset{.5}{\mapsto} \texttt{w} * \texttt{c.fd} \mapsto f * cel_i^*(f, \texttt{mod}{+}i{-}2)}_i\end{array}\right\}$$

```
// But now we have a contradiction, because we have
// two differing views of boxed region 4.

// remove c from list
node* p = (node*) (&start - 1);
while (p->next != c) p = p->next;
```

```
  p->next = c->next;

  // return c to list
  p = (node*) (&start - 1);
  while ((p->next != 0) && (p->next)->val < w) p = p->next;
  c->next = p->next;
  p->next = c;

  // remove c from its old class
  p = (node*) (mod + (v%4) - 2);
  while (p->fd != c) p = p->fd;
  p->fd = c->fd;

  // add c to its new class
  p = (node*) (mod + (w%4) - 2);
  c->fd = p->fd;
  p->fd = c;
}
```

Let us now expand out an instance of these predicates, to see how the double-star and the single star interact. We will demonstrate a possible state that comprises nodes with

values 2,4,7 and 9.

$$
\begin{array}{rcl}
l, s \models state & \Leftrightarrow & l = \{start \mapsto x, mod \mapsto (c_0, c_1, c_2, c_3)\} + l_i + l_c \\
& & \wedge\, l_i, s \models list(x) \\
& & \wedge\, l_c, s \models class_0(c_0) * class_1(c_1) * class_2(c_2) * class_3(c_3) \\
& \Leftrightarrow & l = \{start \mapsto x, mod \mapsto (c_0, c_1, c_2, c_3)\} + l_i + l_0 + l_1 + l_2 + l_3 \\
& & \wedge\, l_i, s \models list(x) \\
& & \wedge\, s = s_0 + s_1 + s_2 + s_3 \\
& & \wedge\, l_0, s_0 \models class_0(c_0) \wedge l_1, s_1 \models class_1(c_1) \\
& & \wedge\, l_2, s_2 \models class_2(c_2) \wedge l_3, s_3 \models class_3(c_3) \\
\hline
l_0, s_0 \models class_0(c_0) & \Leftrightarrow & l_0 = \{c_0.\mathtt{fd} \mapsto c_0'\} + l_0' \\
& & \wedge\, l_0', s_0 \models \boxed{c_0.\mathtt{val} \mapsto v \wedge v \bmod 4 = 0} * list_n(c_0', c_0) \\
& \Leftarrow & l_0 = \{c_0.\mathtt{fd} \mapsto c_0'\} + l_0' \\
& & \wedge\, s_0 = \{c_0.\mathtt{val} \mapsto 4\} + s_0' \\
& & \wedge\, l_0', s_0' \models list_n(c_0', c_0) \\
& \Leftarrow & l_0 = \{c_0.\mathtt{fd} \mapsto c_0'\} + l_0' \\
& & \wedge\, s_0 = \{c_0.\mathtt{val} \mapsto 4\} + s_0' \\
& & \wedge\, l_0', s_0' \models c_0' \doteq c_0 \\
& \Leftrightarrow & l_0 = \{c_0.\mathtt{fd} \mapsto c_0\} \wedge s_0 = \{c_0.\mathtt{val} \mapsto 4\} \\
\hline
l, s \models state & \Leftarrow & l = \{start \mapsto x, mod \mapsto (c_0, c_1, c_2, c_3)\} + l_i + l_0 + l_1 + l_2 + l_3 \\
& & \wedge\, l_i, s \models list(x) \\
& & \wedge\, s = s_0 + s_1 + s_2 + s_3 \\
& & \wedge\, l_0 = \{c_0.\mathtt{fd} \mapsto c_0\} \wedge s_0 = \{c_0.\mathtt{val} \mapsto 4\} \\
& & \wedge\, l_1 = \{c_1.\mathtt{fd} \mapsto c_1\} \wedge s_1 = \{c_1.\mathtt{val} \mapsto 9\} \\
& & \wedge\, l_2 = \{c_2.\mathtt{fd} \mapsto c_2\} \wedge s_2 = \{c_2.\mathtt{val} \mapsto 2\} \\
& & \wedge\, l_3 = \{c_3.\mathtt{fd} \mapsto c_3\} \wedge s_3 = \{c_3.\mathtt{val} \mapsto 7\} \\
& \Leftrightarrow & l = \{start \mapsto x, mod \mapsto (c_0, c_1, c_2, c_3), c_0.\mathtt{fd} \mapsto c_0, c_1.\mathtt{fd} \mapsto c_1, c_2.\mathtt{fd} \mapsto c_2, c_3.\mathtt{fd} \mapsto c_3\} + l_i \\
& & \wedge\, s = \{c_0.\mathtt{val} \mapsto 4, c_1.\mathtt{val} \mapsto 9, c_2.\mathtt{val} \mapsto 2, c_3.\mathtt{val} \mapsto 7\} \\
& & \wedge\, l_i, s \models list(x) \\
\hline
l_i, s \models list_0(x) & \Leftrightarrow & l_i = \{x.\mathtt{next} \mapsto x'\} + l_i' \\
& & \wedge\, l_i', s \models \boxed{x.\mathtt{val} \mapsto v \wedge v \geq n} * list_v(x') \\
& \Leftarrow & l_i = \{x.\mathtt{next} \mapsto x'\} + l_i' \\
& & \wedge\, s = \{x.\mathtt{val} \mapsto 2\} + s' \\
& & \wedge\, l_i', s' \models list_2(x') \\
& \Leftarrow & l_i = \{x.\mathtt{next} \mapsto x', x'.\mathtt{next} \mapsto x''\} + l_i'' \\
& & \wedge\, s = \{x.\mathtt{val} \mapsto 2, x'.\mathtt{val} \mapsto 4\} + s'' \\
& & \wedge\, l_i'', s'' \models list_4(x'') \\
& \Leftarrow & l_i = \{x.\mathtt{next} \mapsto x', x'.\mathtt{next} \mapsto x'', x''.\mathtt{next} \mapsto x'''\} + l_i''' \\
& & \wedge\, s = \{x.\mathtt{val} \mapsto 2, x'.\mathtt{val} \mapsto 4, x''.\mathtt{val} \mapsto 7\} + s''' \\
& & \wedge\, l_i''', s''' \models list_7(x''') \\
& \Leftarrow & l_i = \{x.\mathtt{next} \mapsto x', x'.\mathtt{next} \mapsto x'', x''.\mathtt{next} \mapsto x''', x'''.\mathtt{next} \mapsto x''''\} + l_i'''' \\
& & \wedge\, s = \{x.\mathtt{val} \mapsto 2, x'.\mathtt{val} \mapsto 4, x''.\mathtt{val} \mapsto 7, x'''.\mathtt{val} \mapsto 9\} + s'''' \\
& & \wedge\, l_i'''', s'''' \models list_9(x'''') \\
& \Leftarrow & l_i = \{x.\mathtt{next} \mapsto x', x'.\mathtt{next} \mapsto x'', x''.\mathtt{next} \mapsto x''', x'''.\mathtt{next} \mapsto \bot\} \\
& & \wedge\, s = \{x.\mathtt{val} \mapsto 2, x'.\mathtt{val} \mapsto 4, x''.\mathtt{val} \mapsto 7, x'''.\mathtt{val} \mapsto 9\} \\
\hline
l, s \models state & \Leftarrow & l = \{start \mapsto x, mod \mapsto (c_0, c_1, c_2, c_3), c_0.\mathtt{fd} \mapsto c_0, c_1.\mathtt{fd} \mapsto c_1, c_2.\mathtt{fd} \mapsto c_2, c_3.\mathtt{fd} \mapsto c_3\} + l_i \\
& & \wedge\, s = \{c_0.\mathtt{val} \mapsto 4, c_1.\mathtt{val} \mapsto 9, c_2.\mathtt{val} \mapsto 2, c_3.\mathtt{val} \mapsto 7\} \\
& & \wedge\, l_i = \{x.\mathtt{next} \mapsto x', x'.\mathtt{next} \mapsto x'', x''.\mathtt{next} \mapsto x''', x'''.\mathtt{next} \mapsto \bot\} \\
& & \wedge\, s = \{x.\mathtt{val} \mapsto 2, x'.\mathtt{val} \mapsto 4, x''.\mathtt{val} \mapsto 7, x'''.\mathtt{val} \mapsto 9\} \\
& \Leftrightarrow & l = \{start \mapsto c_2, mod \mapsto (c_0, c_1, c_2, c_3), \\
& & c_0.\mathtt{fd} \mapsto c_0, c_1.\mathtt{fd} \mapsto c_1, c_2.\mathtt{fd} \mapsto c_2, c_3.\mathtt{fd} \mapsto c_3, \\
& & c_2.\mathtt{next} \mapsto c_0, c_0.\mathtt{next} \mapsto c_3, c_3.\mathtt{next} \mapsto c_1, c_1.\mathtt{next} \mapsto \bot\} \\
& & \wedge\, s = \{c_0.\mathtt{val} \mapsto 4, c_1.\mathtt{val} \mapsto 9, c_2.\mathtt{val} \mapsto 2, c_3.\mathtt{val} \mapsto 7\}
\end{array}
$$