* Language design choices (e.g. default values for variables)
* Missing language features (e.g. alloc statement)
* Miscellaneous tool flaws (e.g. do-loop to while-loop)
* Explicit vs implicit variable declarations
* Existentially-bound region in-arguments
* Rule application inference
* Region creation inference
* Region equality inference
* Region assertion duplicability
* Angelic choice
* Invariant region state

|  |  |
| --- | --- |
| primitive\_atomic **procedure** CAS(bag x, node now, node thn) returns (int success)  requires x.hd |-> ?v;  ensures success == 0 || success == 1;  ensures v == now  ? success == 1 && x.hd |-> thn  : success == 0 && x.hd |-> v;  **predicate** Inv(int v)  **struct** bag {  node hd;  id \_nextid;  }  **region** Bag(id r, bag x)  guards { duplicable Z; }  interpretation {  x.hd |-> ?y && x.\_nextid |-> ?y\_r && BagList(y\_r, y, \_, \_, 0) && OWN@y\_r  }  state { 0 }  actions {}  **struct** node {  int value;  node next;  int \_absstate;  id \_nextid;  }  **region** BagList(id r, node y; int v, node z)  guards { unique OWN; }  interpretation {  y.\_absstate |-> ?a &&  y.value |-> ?v &&  y.next |-> ?z &&  y.\_nextid |-> ?z\_r &&  ( a == 0 ? (y != null ==> BagList(z\_r, z, \_, \_, 0) && OWN@z\_r && Inv(v))  : a == 1 ? (BagList(z\_r, z, \_, \_, 0) && OWN@r)  : false)  }  state { a }  actions {  OWN: 0 ~> Set(0, 1);  }    **procedure** push(id r, bag x, int v)  requires Bag(r, x, 0) && Inv(v) && Z@r;  ensures Bag(r, x, 0);  {  node y;  int b;  node t;  id y\_r;  inhale y.value |-> \_;  inhale y.next |-> \_;  y.value := v;  y.next := null;    **do**  invariant Bag(r, x, 0) && Z@r;  invariant b == 0 ==> y.value |-> v && y.next |-> \_ && Inv(v);  {  open\_region  using Bag(r, x);  {  t := x.hd;  }  y.next := t;  use\_atomic  using Bag(r, x) with Z@r;  {  assert x.\_nextid |-> ?t\_r;  b := CAS(x, t, y);  if (b == 1) {  inhale y.\_absstate |-> 0;  inhale y.\_nextid |-> t\_r;  inhale OWN@y\_r;  fold BagList(y\_r, y);  exhale x.\_nextid |-> \_;  inhale x.\_nextid |-> y\_r;  }  }  } **while** (b == 0);  }  **procedure** popCAS(id r, bag x, id t\_r, node t, id t2\_r, node t2) returns (int success)  requires Bag(r, x, 0) && BagList(t\_r, t, ?v, t2) && BagList(t2\_r, t2) && t != null && Z@r;  ensures success == 0 || success == 1;  ensures success == 1 ==> Inv(v);  ensures Z@r;  {  use\_atomic  using Bag(r, x) with Z@r;  {  success := CAS(x, t, t2);  **if** (success == 1) {  exhale x.\_nextid |-> ?t\_r0;  inhale x.\_nextid |-> t2\_r;  assert BagList(t\_r0, t, ?v0, ?t2\_0, 0);  assume t\_r0 == t\_r && v0 == v && t2\_0 == t2;  use\_atomic  using BagList(t\_r, t) with OWN@t\_r;  {  assert t.\_nextid |-> ?t2\_r0;  assume t2\_r0 == t2\_r;  exhale t.\_absstate |-> 0;  inhale t.\_absstate |-> 1;  }  }  }  }  **procedure** pop(id x\_r, bag x) returns (int status, int ret)  requires Bag(x\_r, x, 0) && Z@r;  ensures status == 1 ? Bag(x\_r, x, 0) && Inv(ret) : Bag(x\_r, x, 0);  ensures status == -1 || status == 1;  {  node t;  node t2;  id t\_r;  id t2\_r;  int v;  status := 0;  exhale status == 1 ==> BagList(t\_r, t, v, t2, \_) && t != null && Inv(v);  **while** (status == 0)  invariant -1 <= status <= 1;  invariant Bag(x\_r, x, 0) && Z@r;  {  inhale status == 1 ==> BagList(t\_r, t, v, t2, \_) && t != null && Inv(v);  open\_region  using Bag(x\_r, x);  {  t := x.hd;  assert x.\_nextid |-> ?x\_nid1;  havoc t\_r;  assume t\_r == x\_nid1;  assert BagList(t\_r, t, ?t\_v);  inhale BagList(t\_r, t);  havoc v;  assume v == t\_v;  }  **if** (t == null) {  status := -1;  } **else** {  open\_region  using BagList(t\_r, t);  {  t2 := t.next;  assert t.\_nextid |-> ?t\_nid1;  havoc t2\_r;  assume t2\_r == t\_nid1;  assert BagList(t2\_r, t2);  inhale BagList(t2\_r, t2);  }  assert BagList(t\_r, t, ?v\_1, ?t2\_1);  assume v\_1 == v && t2\_1 == t2;  assert Bag(x\_r, x, 0) && BagList(t\_r, t, v, t2) && BagList(t2\_r, t2);  inhale Bag(x\_r, x, 0) && BagList(t\_r, t, v, t2) && BagList(t2\_r, t2);  status := popCAS(x\_r, x, t\_r, t, t2\_r, t2);  assert BagList(t\_r, t, ?v\_2, ?t2\_2);  assume v\_2 == v && t2\_2 == t2;  exhale status == 1 ==> BagList(t\_r, t, v, t2, \_) && t != null && Inv(v);  }  }  inhale status == 1 ==> BagList(t\_r, t, v, t2, \_) && t != null && Inv(v);  if (status == 1) {  open\_region  using BagList(t\_r, t);  {  ret := t.value;  }  }  } | **predicate** bagInvariant(v);  **region** Bag(r,x) {  guards 0;  interpretation {  0 : x |-> head &\*& BagList(bl,head,\_,\_,0) &\*& bl@OWN;  }  actions {}  }  **region** BagList(s,y,val,z) {  guards OWN;  interpretation {  0 : y = 0 ? true : y |-> val &\*& (y + 1) |-> z &\*& BagList(nxtbl,z,\_,\_,0) &\*& nxtbl@OWN &\*& bagInvariant(val);  1 : s@OWN &\*& y |-> val &\*& (y + 1) |-> z &\*& BagList(nxtbl,z,\_,\_,\_);  }  actions {  OWN : 0 ~> 1;  }  }  **function** push(x,v)  requires Bag(r,x,0) &\*& bagInvariant(v);  ensures Bag(r,x,0);  {            y := alloc(2);    [y] := v;          **do** {        t := [x];            [y + 1] := t;    cr := CAS(x,t,y);                            }  invariant Bag(r,x,0) &\*& (cr = 0 ? y |-> v &\*& y+1 |-> \_ &\*& bagInvariant(v) : true);  **while** (cr = 0);  }  **function** popCAS(x,t,t2)  requires Bag(r,x,0) &\*& BagList(rt,t,v,t2,\_) &\*& BagList(rt2,t2,\_,\_,\_) &\*& t != 0;  ensures ret = 0 \/ bagInvariant(v);  {  cr := CAS(x,t,t2);  return cr;                                  }  **function** pop(x)  requires Bag(r,x,0);  ensures ret = 0 ? Bag(r,x,0) : Bag(r,x,0) &\*& bagInvariant(ret);  {                  **do** {              t := [x];                          **if** (t = 0) {  return 0;  }  t2 := [t + 1];                      cr := popCAS(x,t,t2);            }  invariant Bag(r,x,0) &\*& (cr = 0 ? true : BagList(rt,t,v,t2,\_) &\*& t != 0 &\*& bagInvariant(v));  **while** (cr = 0);    ret := [t];  return ret;            } |