- 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);
  }
```

```
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;
  }
}
```

```
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);
}
```

```
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);
}
```

```
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;
      }
    }
  }
}
```

```
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;









}
```

```
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);
```

```
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);
    }
  }
  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;
    }
  }
}
```

```
  }
          invariant Bag(r,x,0) &*& (cr = 0 ? true : BagList(rt,t,v,t2,_) &*& t != 0
&*& bagInvariant(v));
        while (cr = 0);

  ret := [t];
  return ret;





}
```