在 Coq 中使用分离逻辑

C 语言数据结构中的单链表：

```
struct list {
   int data;
   struct list *next;
};
```

单链表的表示谓词：

```
Fixpoint sll (x: addr) (l: list Z): Assertion :=
  match l with
    | nil     => [| x = NULL |] && emp
    | a :: l0 => [| x <> NULL |] &&
                 EX y: addr,
                   &(x # "list" ->  "data") # Int |-> a **
                   &(x # "list" ->  "next") # Ptr |-> y **
                   sll y l0
  end.
```

基于 `sll` 证明 `reverse` 函数：

```
struct list *reverse(struct list *p)
/*@ With (l: list Z)
    Require sll(p, l)
    Ensure sll(__return, rev(l))
*/
{
   struct list * w = ( void * ) 0, * v = p;
   /*@ Inv exists l1 l2,
            l == app(rev(l1), l2) &&
            sll(w, l1) * sll(v, l2)
      */
   while (v) {
      struct list * t = v -> next;
      v -> next = w;
      w = v;
      v = t;
   }
   return w;
}
```

用分离逻辑描述单链表的一段：

```
Fixpoint sllseg (x y: addr) (l: list Z): Assertion :=
  match l with
    | nil     => [| x = y |] && emp
    | a :: l0 => [| x <> NULL |] &&
                 EX z: addr,
                   &(x # "list" ->  "data") # Int |-> a **
                   &(x # "list" ->  "next") # Ptr |-> z **
                   sllseg z y l0
  end.
```

`append` 的验证：

```
struct list *append(struct list *x, struct list *y)
/*@ With l1 l2
    Require sll(x, l1) * sll(y, l2)
    Ensure sll(__return, app(l1, l2))
*/
{
  struct list *t, *u;
  if (x == ( void * )0) {
    return y;
  } else {
    t = x;
    u = t -> next;
    /*@ Inv
          exists l1a l1b,
            app(l1a, cons(t -> data, l1b)) == l1 &&
            t != 0 && t -> next == u &&
            sllseg(x, t, l1a) *
            sll(u, l1b) * sll(y, l2)
    */
    while (u != ( void * )0) {
      t = u;
      u = t -> next;
    }
    t -> next = y;
    return x;
  }
}
```