A CONTAINER LIBRARY FOR HI-LITE

Content

- A container library adapted to specification
- An axiomatization for formal proof
- A validation using a proof assistant

A CONTAINER LIBRARY ADAPTED TO SPECIFICATION

Our running example

```
procedure Map_F (L : in out List) is
  Current : Cursor := First (L);
begin
  while Current /= No_Element loop
    Replace_Element
        (L, Current,
         F (Element (Current)));
    Next (Current);
  end loop;
end Map_F;
```

Container Types

```
procedure Map_F (L : in out List) is
  Current : Cursor := First (L);
begin
  while Current /= No_Element loop
    Replace_Element
        (L, Current,
         F (Element (Current)));
    Next (Current);
  end loop;
end Map_F;
```

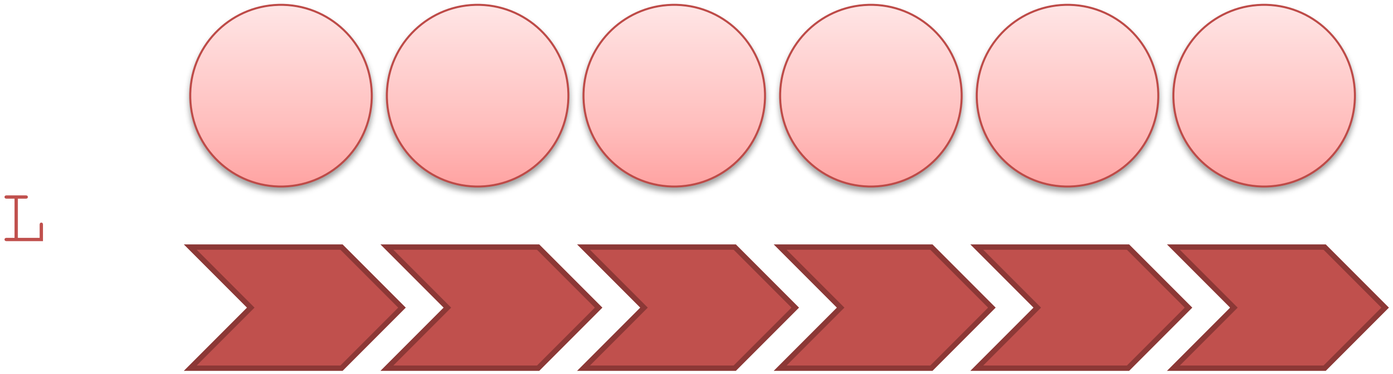
Iteration through cursors

```
procedure Map_F (L : in out List) is
  Current : Cursor := First (L);
begin
  while Current /= No_Element loop
    Replace_Element
        (L, Current,
         F (Element (Current)));
    Next (Current);
  end loop;
end Map_F;
```

Modification

```
procedure Map_F (L : in out List) is
  Current : Cursor := First (L);
begin
  while Current /= No_Element loop
    Replace_Element
        (L, Current,
         F (Element (Current)));
    Next (Current);
  end loop;
end Map_F;
```

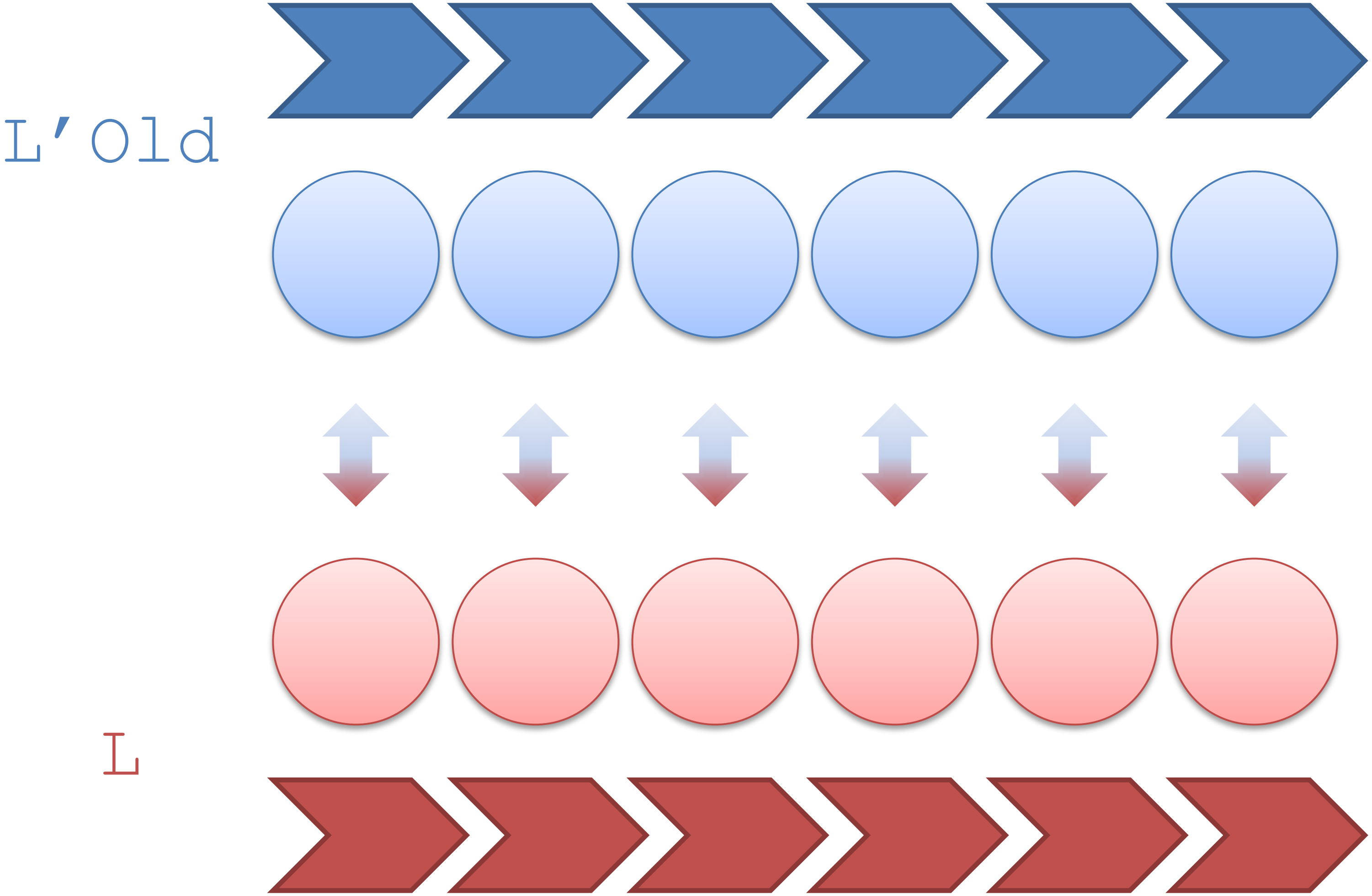
A List



Modification

```
procedure Map_F (L : in out List) is
  Current : Cursor := First (L);
begin
  while Current /= No_Element loop
    Replace_Element
        (L, Current,
         F (Element (Current)));
    Next (Current);
  end loop;
end Map_F;
```

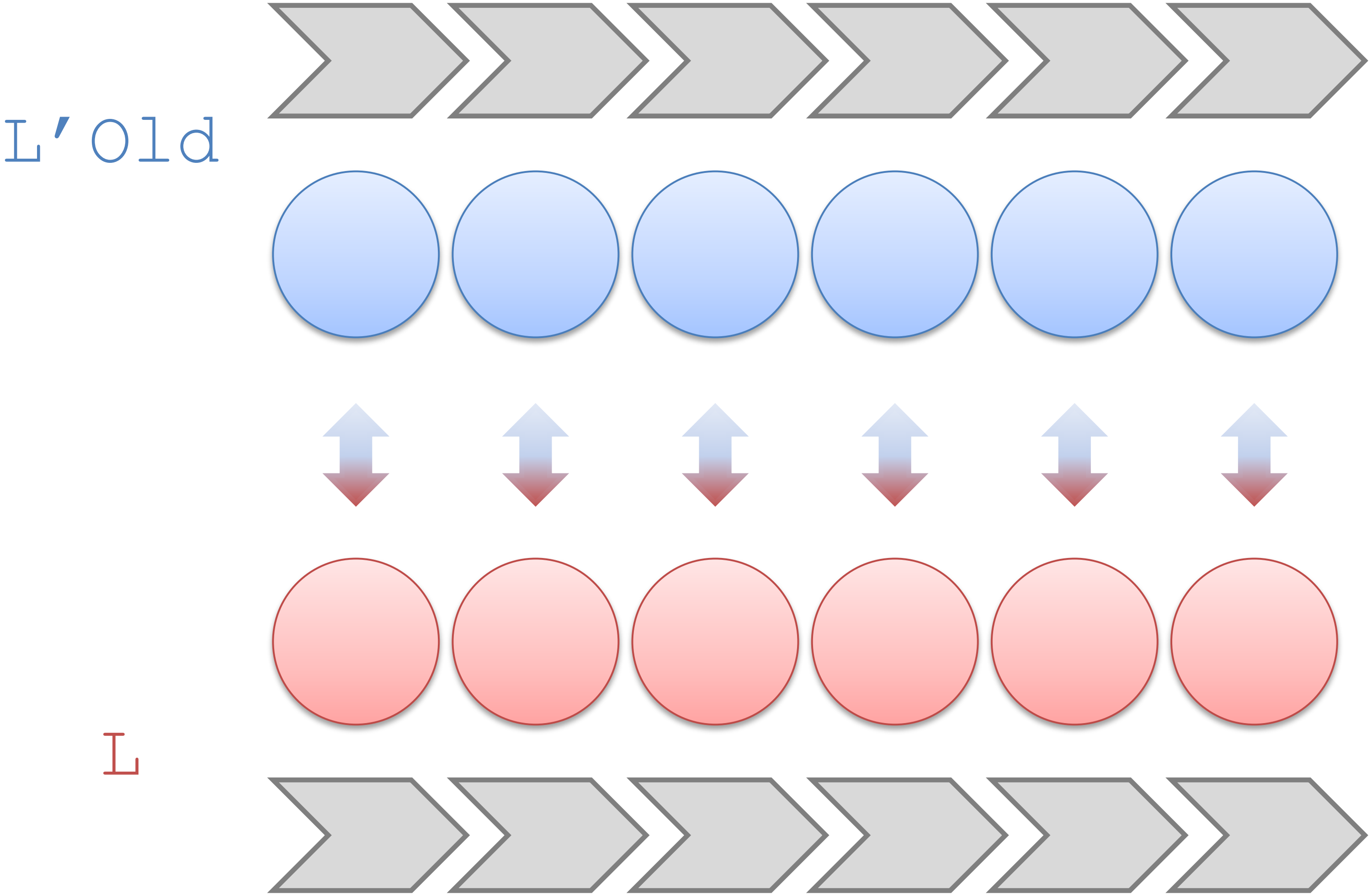
Specify `Map_F`



With quantified expressions

```
procedure Map_F
             (L : in out List)
with
  Post =>
    (for all Cu in L =>
       Element (Cu) =
        F (Element (   )))
```

On independent cursors



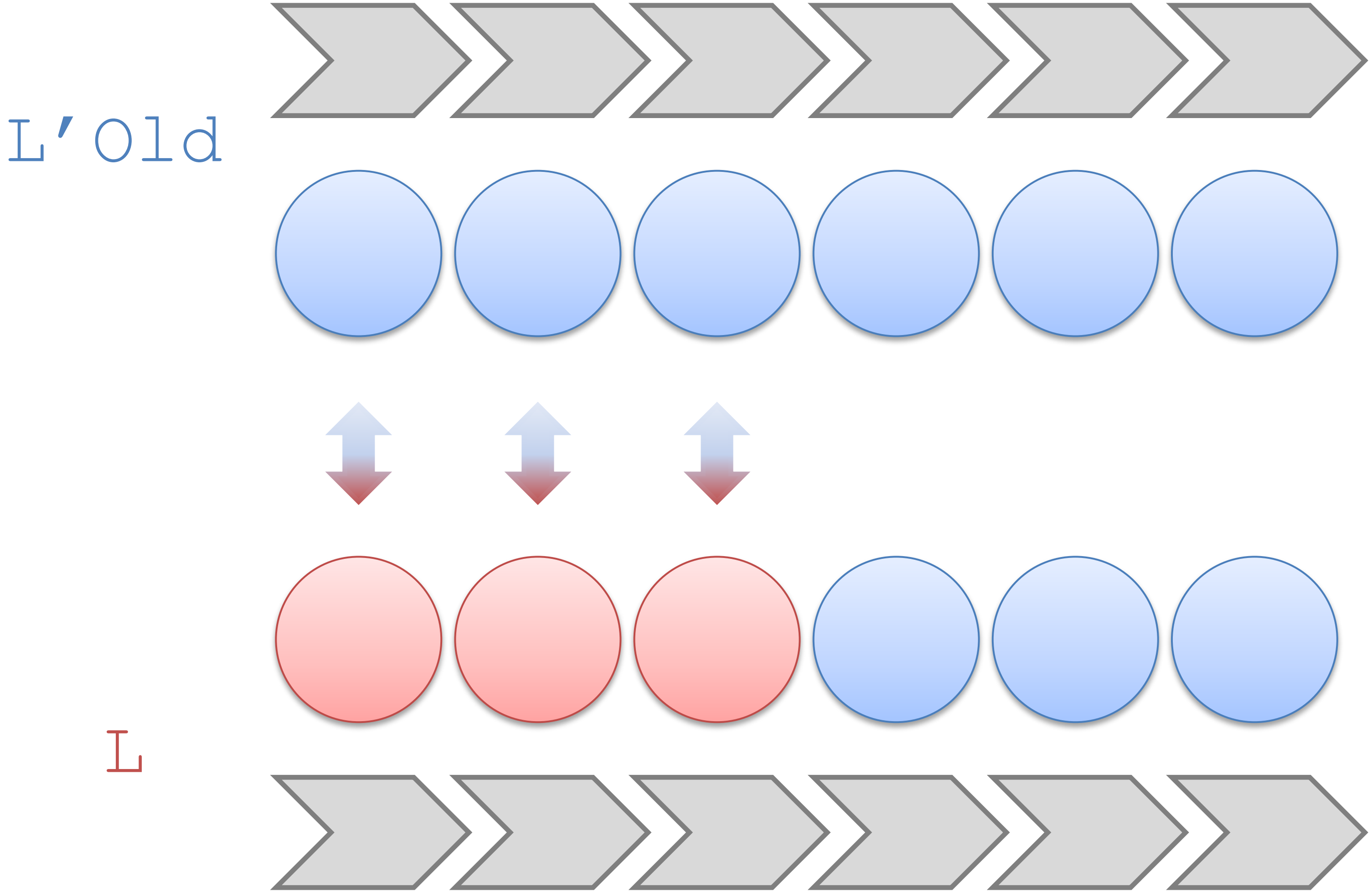
Map_F's Contract

```
procedure Map_F
           (L : in out List)
with
  Post =>
    (for all Cu in L =>
       Element (L, Cu) =
        F (Element (L'Old, Cu)))
```

For the loop invariant

```
procedure Map_F (L : in out List) is
   Current : Cursor := First (L);
begin
   while Current /= No_Element loop
      Replace_Element
           (L, Current,
            F (Element (L, Current)));
      Next (L, Current);
   end loop;
end Map_F;
```

Use part of containers



`Map_F`’s loop invariant

```
(for all Cu in Left (L, Current)
  =>
   Element (L, Cu) =
      F (Element (L’Old, Cu)))
and
   Strict_Equal
     (Right (L, Current),
      Right (L’Old, Current)))
```

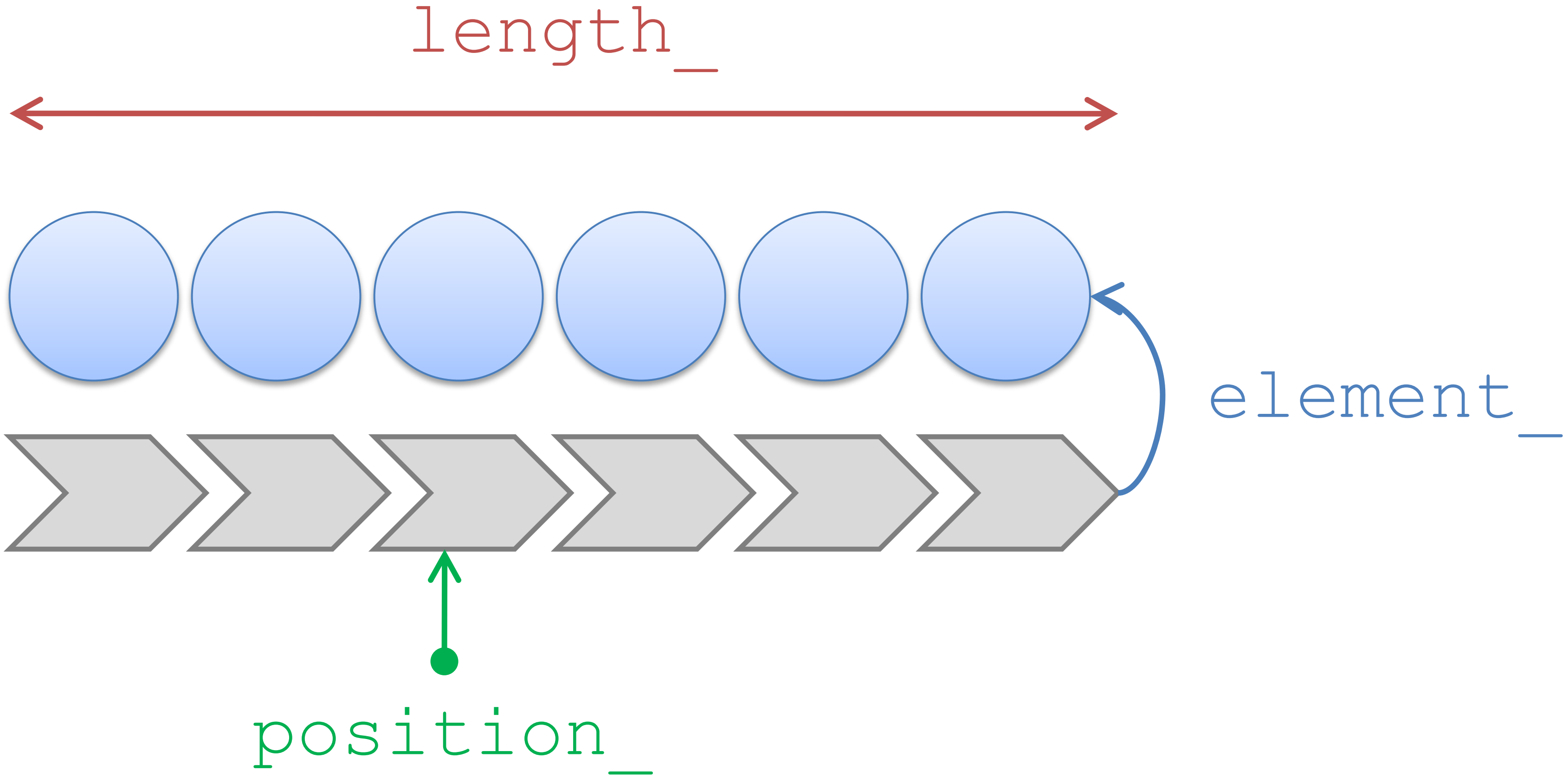
AN AXIOMATIZATION FOR FORMAL PROOF

Read description from RM

```
procedure Replace_Element
    (Container : in out List;
     Position  : in     Cursor;
     New_Item  : in     Element_Type);
```

"If Position does not designate an element in Container, then Program_Error is propagated. Otherwise Replace_Element assigns the value New_Item to the element designated by Position."

Define logic functions


`length_`

`element_`

`position_`

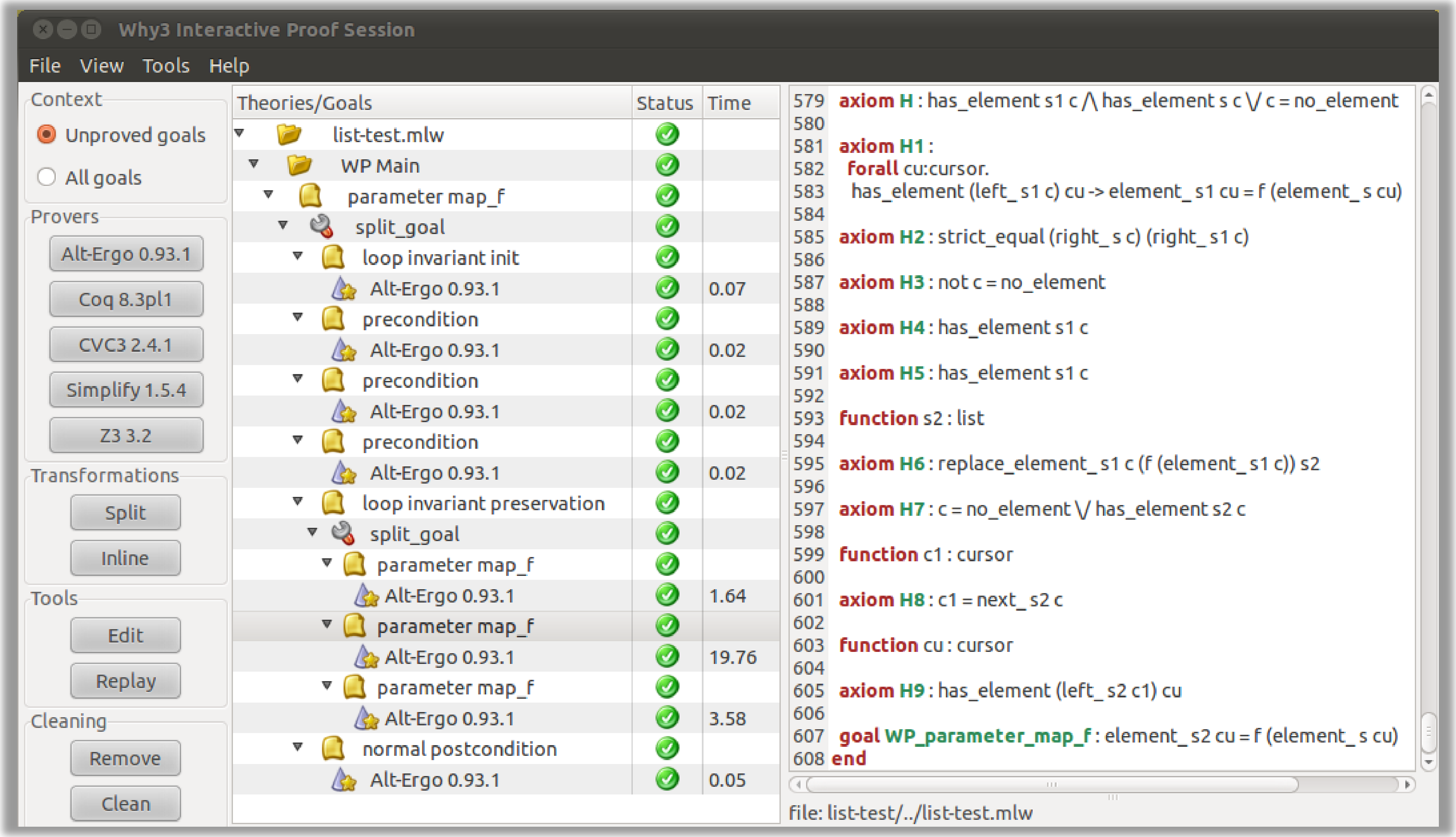
Used in contract

```
val replace_element :
    l : ref list  -> cu : cursor ->
    e : element_t ->
  {position_ !l cu > 0              }
    unit writes l
  {replace_element_ (old !l) cu e !l}
```

Formally describe effects

```
element_ !l cu = e and
length_ !l = length_ (old !l) and
(forall cun : cursor.
  position_ !l cun =
  position_ (old !l) cun) and
(forall cun : cursor.
  cu <> cun and
  position_ !l cun > 0 ->
    element_ !l cun =
    element_ (old !l) cun)
```

Automatically verify function



Theories/Goals	Status	Time
list-test.mlw	✓	
WP Main	✓	
parameter map_f	✓	
split_goal	✓	
loop invariant init	✓	
Alt-Ergo 0.93.1	✓	0.07
precondition	✓	
Alt-Ergo 0.93.1	✓	0.02
precondition	✓	
Alt-Ergo 0.93.1	✓	0.02
precondition	✓	
Alt-Ergo 0.93.1	✓	0.02
loop invariant preservation	✓	
split_goal	✓	
parameter map_f	✓	
Alt-Ergo 0.93.1	✓	1.64
parameter map_f	✓	
Alt-Ergo 0.93.1	✓	19.76
parameter map_f	✓	
Alt-Ergo 0.93.1	✓	3.58
normal postcondition	✓	
Alt-Ergo 0.93.1	✓	0.05

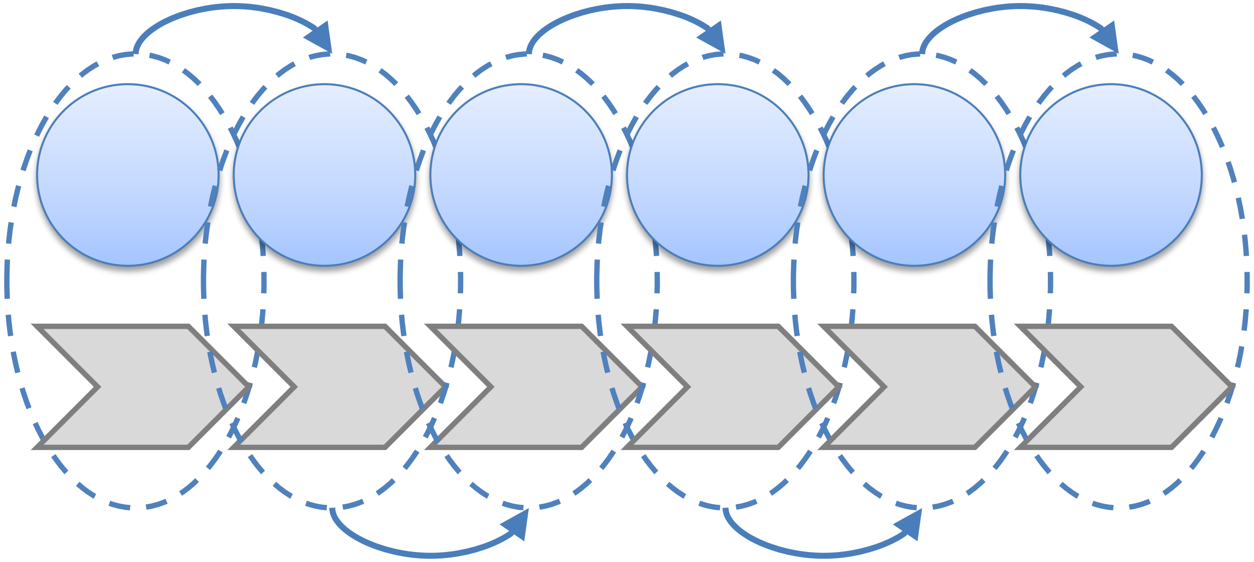
```
axiom H : has_element s1 c /\ has_element s c \/ c = no_element

axiom H1 :
 forall cu:cursor.
  has_element (left_ s1 c) cu -> element_ s1 cu = f (element_ s cu)

axiom H2 : strict_equal (right_ s c) (right_ s1 c)

axiom H3 : not c = no_element

axiom H4 : has_element s1 c

axiom H5 : has_element s1 c

function s2 : list

axiom H6 : replace_element_ s1 c (f (element_ s1 c)) s2

axiom H7 : c = no_element \/ has_element s2 c

function c1 : cursor

axiom H8 : c1 = next_ s2 c

function cu : cursor

axiom H9 : has_element (left_ s2 c1) cu

goal WP_parameter_map_f : element_ s2 cu = f (element_ s cu)
end
```

file: list-test/../list-test.mlw

A VALIDATION USING A PROOF ASSISTANT

Define a representation

```
Definition Rlist : Set :=
    List.list (cursor*element_t)
```



Implement logic functions

```
Fixpoint position (l : Rlist)
 (cu : cursor) (n : nat) : nat :=
   match l with
       nil      => 0
     | a :: ls =>
        if beq_nat (fst a) cu
        then n
        else position ls cu (S n)
   end.
```

Implement functions' description

```
Fixpoint replace
 (l : Rlist) (cu : cursor)
 (e : element_t) : Rlist :=
  match l with
      nil      => nil
    | a :: ls =>
       if beq_nat (fst a) cu
       then (fst a, e) :: ls
       else a :: (replace ls cu e)
  end.
```

Prove functions' contracts

```
Lemma replace_length :
  forall l  :     Rlist,
  forall cu :    cursor,
  forall e  : element_t,
  position l cu 1 > 0 ->
    length l =
      length (replace l cu e).
```

Conclusion

- An API for imperative containers
- Adapted to specification process
- With executable annotations

- An axiomatization of these containers
- Based on the manual specifications
- Validated through a model in Coq