Preuve formelle de micro-noyau

Adrien Stalain

May 13, 2019

```
theory Listp
  imports "Import_C"
begin
```

1 list

1.1 list definition

Le `list_C` est un type importé du C (struct list) qui représente une liste chainée, sa définition est:

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

```
type_synonym node = "list_C ptr"
```

```
primrec list :: "node list ⇒ node ⇒ lifted_globals ⇒ bool"  where
list_is_empty:  "list [] p s = (p = NULL)" |
list_is_cons:  "list (x#xs) p s = ( p = x
                                ∧ is_valid_list_C s p
                                ∧ p ≠ NULL
                                ∧ list xs s[p]→next s
                                )"
```

La propritétée `list x p s` est vraie quand `p` est une liste valide contenant les nodes `x` dans l'état global `s`

Une liste bien définie est soit vide, et dans ce cas là `p` est `NULL`, soit une liste commençant par `x` et avec comme reste `xs` et dans ce cas là `p = x`, p est valide dans l'état de heap `s` (`is_valid_list_C s p`) `p` n'est pas `NULL`, et `xs` est une liste valide qui commence par `s[p]→next`

```
definition a_list :: "node ⇒ lifted_globals ⇒ bool" where
  "a_list p s ≡ (∃xs. list xs p s)"
```

La propriétée `a_list p s` est valide si `p` est une liste valide dans l'état `s`

1.2 list properties

```
lemma list_is_empty_r : "list a NULL s ⟹ [] = a"
```

Une liste valide qui commence par `NULL` ne peux que être vide

lemma list_is_cons_r : "p $\neq$ NULL $\Longrightarrow$ list x p s = ($\exists$ys. (x = p#ys) $\wedge$ (is_valid_list_C s p)
$\wedge$ (p $\neq$ NULL) $\wedge$ (list ys s[p]$\rightarrow$next s))"

Une liste valide qui n'est pas NULL, contient au moins un élément

lemma list_not_2_same : "list (x#y#z) p s $\Longrightarrow$ x $\neq$ y"

lemma list_append_Ex: "list (xs @ ys) p s $\Longrightarrow$ ($\exists$q. list ys q s)"

lemma list_unique: "$\llbracket$ list xs p s ; list ys p s $\rrbracket$ $\Longrightarrow$ xs = ys"

lemma list_distinct : "list x p s $\Longrightarrow$ distinct x"

lemma list_head_not_in_cons : "list (x#xs) x s $\Longrightarrow$ x $\notin$ set xs"

lemma the_list_unique : "list xs p s $\Longrightarrow$ (THE ys. list ys p s) = xs"

lemma list_next_in_list : "$\llbracket$ list xs p s ; a $\in$ set xs ; s[a]$\rightarrow$next $\neq$ NULL $\rrbracket$$\Longrightarrow$(s[a]$\rightarrow$next) $\in$ set xs"

lemma list_has_end_not_null : "list (xs @ [x]) p s $\Longrightarrow$ p $\neq$ NULL"

lemma list_no_next_is_last : "$\llbracket$ list (xs @ [x]) p s ; w $\in$ set (xs @ [x]) ; s[w]$\rightarrow$next = NULL $\rrbracket$$\Longrightarrow$ w = x"

lemma list_last_next_is_null : "list (xs @ [x]) p s $\Longrightarrow$ s[x]$\rightarrow$next = NULL"

lemma list_last_next_is_null2 : "$\llbracket$ xs $\neq$ [] ; list xs p s $\rrbracket$ $\Longrightarrow$ s[last xs]$\rightarrow$next = NULL"

lemma list_content_is_valid : "$\llbracket$ list xs p s ; w $\in$ set xs $\rrbracket$ $\Longrightarrow$ is_valid_list_C s w $\wedge$ w $\neq$ NULL
$\wedge$ ($\exists$ys. list ys s[w]$\rightarrow$next s)"

lemma first_element_in_list : "$\llbracket$ list xs p s ; p $\neq$ NULL $\rrbracket$ $\Longrightarrow$ p $\in$ set xs"

lemma a_list_next_is_a_list : "$\llbracket$ p $\neq$ NULL ; a_list p s $\rrbracket$ $\Longrightarrow$ a_list s[p]$\rightarrow$next s"

1.2.1 state update

lemma list_st_update_ignore [simp] : "q $\notin$ set xs $\Longrightarrow$ list xs p (s[q$\rightarrow$next := ω]) = list xs p s"

lemma list_st_add [simp] : "$\llbracket$ is_valid_list_C s x ; x $\neq$ NULL; x $\notin$ set xs $\rrbracket$
$\Longrightarrow$ list (x#xs) x s[x$\rightarrow$next := p] = list xs p s"

lemma list_st_upd_any_base_ptr [simp] : "ptr_coerce (l :: list_C ptr ptr) $\notin$ set xs
$\Longrightarrow$ list xs p
s[l := ω] = list xs p s"

2 listp

2.1 listp definition

definition listp :: "node list $\Rightarrow$ node ptr $\Rightarrow$ lifted_globals $\Rightarrow$ bool" **where**
"listp n pt s $\equiv$ (ptr_coerce pt $\notin$ set n $\wedge$ is_valid_list_C'ptr s pt $\wedge$ list n s[pt] s)"

La propriétée listp x p s indique que p est une liste valide contenant les nodes x dans l'état global s

2.2 listp properties

lemma listp_unique: "⟦ listp xs p s ; listp ys p s ⟧ $\Longrightarrow$ xs = ys"

3 list_length

3.1 list_length definition

primrec list_length :: "nat $\Rightarrow$ list_C ptr $\Rightarrow$ lifted_globals $\Rightarrow$ bool" **where**
list_length_empty : "list_length 0 p s = (p = NULL)" |
list_length_suc : "list_length (Suc n) p s = (p $\neq$ NULL $\wedge$ is_valid_list_C s p $\wedge$ list_length n s[p]$\rightarrow$next s)"

definition list_length_p :: "nat $\Rightarrow$ list_C ptr ptr $\Rightarrow$ lifted_globals $\Rightarrow$ bool" **where**
"list_length_p n p s $\equiv$ is_valid_list_C'ptr s p $\wedge$ list_length n s[p] s"

3.2 list_length properties

lemma list_length_empty_r : "list_length n NULL s $\Longrightarrow$ n = 0"

lemma list_length_suc_r : "p $\neq$ NULL $\Longrightarrow$ list_length n p s = ($\exists$ns. is_valid_list_C s p $\wedge$ Suc ns = n $\wedge$ list_length ns s[p]$\rightarrow$next s)"

lemma list_length_has_list : "list_length n p s $\Longrightarrow$ ($\exists$xs. list xs p s $\wedge$ length xs = n)"

lemma list_length_unique : "⟦ list_length n p s $\wedge$ list_length m p s ⟧ $\Longrightarrow$ n = m"

lemma list_length_equiv_the_length_list : "list_length n p s $\Longrightarrow$ length (THE xs. list xs p s) = n"

lemma list_length_equiv_length_list2 : "list xs p s $\Longrightarrow$ list_length (length xs) p s"

lemma list_length_equiv_length_list3 : "⟦ list xs p s ; list_length x p s ⟧ ⟹ x = length xs"

lemma list_length_equiv_length_list4 : "list_length x p s ⟹ (THE n. list_length n p s) = x"

lemma list_length_p_is_list_length : "list_length_p n p s ⟹ list_length n s[p] s"

lemma list_length_the_is_zero : "b = NULL ⟹ (THE n. list_length n b s) = 0"

lemma the_list_length_exists : "list_length m b s ⟹ (THE n. list_length n b s) = m"

lemma list_length_exists : "a_list p s ⟹ ∃n. list_length n p s"

lemma list_length_the_is_not_zero : "⟦ list_length n b s ; b ≠ NULL⟧ ⟹ Suc (THE n. list_length n s[b]→next s) = n"

lemma list_length_the_list_length : "a_list p s ⟹ list_length (THE n . list_length n p s) p s"

lemma list_length_non_null_not_zero : "⟦ a_list p s ; p ≠ NULL ⟧ ⟹ (THE n. list_length n p s) ≠ 0"

4 hoare helpers

4.1 nondet monad

lemma grab_asm_NF : "(G ⟹ ⦃P⦄ f ⦃Q⦄!) ⟹ ⦃λs. G ∧ P s⦄ f ⦃Q⦄!"

lemma hoare_conjINF:
"⟦ ⦃P⦄ f ⦃Q⦄; ⦃P⦄ f ⦃R⦄! ⟧ ⟹ ⦃P⦄ f ⦃λr s. Q r s ∧ R r s⦄!"

lemma hoare_conjINFR:
"⟦ ⦃P⦄ f ⦃Q⦄!; ⦃P⦄ f ⦃R⦄ ⟧ ⟹ ⦃P⦄ f ⦃λr s. Q r s ∧ R r s⦄!"

lemma hoare_transf_predNF: "⟦ (G' ⟹ ⦃ P ⦄ f ⦃ Q ⦄!) ; (G ⟹ G') ⟧ ⟹ ⦃ λs. G ∧ P s ⦄ f ⦃ Q ⦄!"

lemma hoare_transf_pred: "⟦ (G' ⟹ ⦃ P ⦄ f ⦃ Q ⦄) ; (G ⟹ G') ⟧ ⟹ ⦃ λs. G ∧ P s ⦄ f ⦃ Q ⦄"

4.2 option monad

lemma ovalidNF_is_validNF : "ovalidNF P f P' ⟹ ⦃ P ⦄ gets_the f ⦃ P'⦄!"

lemma ovalidNF_is_valid : "ovalidNF P f P' ⟹ ⦃ P ⦄ gets_the f ⦃ P'⦄"

lemma ovalid_is_valid : "ovalid P f P' ⟹ ⦃ P ⦄ gets_the f ⦃ P'⦄"

lemma ovalid_herit_NF : "⟦ ovalid P f Q ; ovalidNF P f Q' ⟧ ⟹ ovalidNF P f Q"

lemma ovalidNF_comb3p [wp_comb]:
"⟦ ovalidNF P f Q; ovalidNF P f Q' ⟧ ⟹ ovalidNF (λs. P s) f (λr s. Q r s ∧ Q' r s)"

lemma ovalidNF_comb3pr [wp_comb]:
"⟦ ovalidNF P f Q; ovalid P f Q' ⟧ ⟹ ovalidNF (λs. P s) f (λr s. Q r s ∧ Q' r s)"

lemma ovalid_grab_asm2: "(P' ⟹ ovalid (λs. P s ∧ R s) f Q) ⟹ ovalid (λs. P s ∧ P' ∧ R s) f Q"

lemma ovalid_drop_post: "⟦ R ; ovalid P f Q ⟧ ⟹ ovalid P f (λr s. R ∧ Q r s)"

lemma ovalidNF_drop_post: "⟦ R ; ovalidNF P f Q ⟧ ⟹ ovalidNF P f (λr s. R ∧ Q r s)"

5 isabelle lists helpers

lemma list_non_empty_has_init : "x ≠ [] ⟹ ∃w. w @ [last x] = x"

6 state helpers

lemma lhu:
"heap_list_C_update f (heap_list_C_update g s) ≡ heap_list_C_update (λh. f (g h)) s"

lemma next_upd_diff : "x ≠w ⟹ s[x→next := a][w→next := b] ≡ s[w→next := b][x→next := a]"

lemma listptrptr_upd_not_mod : "ptr_coerce (k :: list_C ptr ptr) ≠ (x :: list_C ptr)
⟹
s[k := ω][x] = s[x]"

7 program proof

7.1 list_empty

7.1.1 correct

lemma list_empty_correct : "⦃ λ s. is_valid_list_C'ptr s l ⦄ all.list_empty' l ⦃ λ_. listp [] l ⦄!"

7.1.2 pure

lemma `list_empty_pure : "(∀s. P s ⟶ P s[l := NULL]) ⟹ ⦃ P ⦄ all.list_empty' l ⦃ λ_. P ⦄"`

7.1.3 alt

lemma `list_empty_alt1_correct : "⦃ λ s. is_valid_list_C'ptr s l ⦄ all.list_empty_alt1' l ⦃ λ_. listp [] l ⦄!"`

lemma `list_empty_alt1_pure : "(∀s. P s ⟶ P s[l := NULL]) ⟹ ⦃ P ⦄ all.list_empty_alt1' l ⦃ λ_. P ⦄"`

lemma `list_empty_alt2_correct : "⦃ λ s. is_valid_list_C'ptr s l ⦄ all.list_empty_alt2' l ⦃ λ_. listp [] l ⦄!"`

lemma `list_empty_alt2_pure : "(∀s. P s ⟶ P s[l := NULL]) ⟹ ⦃ P ⦄ all.list_empty_alt2' l ⦃ λ_. P ⦄"`

7.2 list_insert_front

7.2.1 bad spec

lemma `list_insert_front_correct_bad_spec : "x ≠ NULL`
`⟹ ⦃λs. listp xs l s ∧ is_valid_list_C s x ⦄ all.list_insert_front' l x ⦃λr. listp (x#xs) l ⦄!"`

7.2.2 correct

lemma `list_insert_front_correct : "⟦ x ∉ set xs ; x ≠ NULL ; x ≠ ptr_coerce l ⟧`
`⟹ ⦃λs. listp xs l s ∧ is_valid_list_C s x ⦄ all.list_insert_front' l x ⦃λr. listp (x#xs) l ⦄!"`

7.2.3 pure

lemma `list_insert_front_pure : "(∀s. P s ⟶ P s[x→next := s[l]][l := x])`
`⟹ ⦃ P ⦄ all.list_insert_front' l x ⦃λ_. P ⦄"`

7.3 list_singleton

7.3.1 correct

lemma `list_singleton_correct : "⟦ x ≠ NULL ; x ≠ ptr_coerce l ⟧ ⟹`
`⦃ λs. is_valid_list_C s x ∧ is_valid_list_C'ptr s l ⦄`
`all.list_singleton' l x`
`⦃ λ r. listp [x] l ⦄!"`

7.3.2 pure

```
lemma list_singleton_pure : "(∀s. P s ⟶ P s[l := x][x→next := NULL])
⟹ ⦃ P ⦄ all.list_singleton' l x ⦃ λ_. P ⦄"
```

7.3.3 bad

```
lemma list_singleton_bad_correct : "⟦ x ≠ NULL ; x ≠ ptr_coerce l ⟧ ⟹

                                                   ⦃ λs. is_valid_list_C s x ∧
is_valid_list_C'ptr s l  ⦄
                                                   all.list_singleton_alt' l x
                                                   ⦃ λ r. listp [x] l ⦄!"
```

7.4 list_insert_after

7.4.1 correct

```
lemma list_insert_inside : "⟦ n ∉ set(x1 @ w # x2) ; n ≠ NULL ; is_valid_list_C
s n
                              ; list (x1 @ [w] @ x2) p s⟧
                              ⟹ list (x1 @ w # n # x2) p s[w→next := n][n→next
:= s[w]→next]"
```

```
lemma list_insert_after_correct : "⟦ x ≠ w ; x ∉ set xa ; x ∉ set xb ; x
≠ NULL ⟧
                                              ⟹ ⦃ λs. list (xa @ [w] @ xb) p
s ∧ is_valid_list_C s x ⦄
                                                   all.list_insert_after' w x
                                                   ⦃ λr s. list (xa @ [w,x] @ xb)
p s ⦄!"
```

```
lemma list_insert_after_correct_p : "⟦ ptr_coerce p ≠ x ;  x ∉ set (xa @
[w] @ xb) ; x ≠ NULL ⟧ ⟹ ⦃ λs. listp (xa @ [w] @ xb) p s ∧ is_valid_list_C
s x ⦄ all.list_insert_after' w x ⦃ λr s. listp (xa @ [w,x] @ xb) p s ⦄!"
```

7.4.2 pure

```
lemma list_insert_after_pure : "(∀s. P s ⟶ P s[x→next := s[w]→next][w→next
:= x]) ⟹ ⦃ P ⦄ all.list_insert_after' w x ⦃ λr. P ⦄"
```

7.4.3 specialisations

```
lemma list_insert_after_last : "⟦ (∃a. a @ w # [] = xs) ; x ∉ set xs ; x
≠ NULL ; (ptr_coerce p) ≠ x ⟧ ⟹ ⦃ λs. listp xs p s ∧ is_valid_list_C s
x ⦄ all.list_insert_after' w x ⦃λ r. listp (xs @ [x]) p ⦄!"
```

lemma list_insert_after_the_last : "⟦ x ∉ set xs ; x ≠ NULL ; (ptr_coerce p) ≠ x ; xs ≠ [] ⟧ ⟹ ⦃λs. listp xs p s ∧ is_valid_list_C s x ⦄ all.list_insert_after' (last xs) x ⦃ λ_. listp (xs @ [x]) p⦄!"

lemma list_insert_after_the_last_pre : "⟦ x ∉ set xs ; x ≠ NULL ; (ptr_coerce p) ≠ x ; xs ≠ [] ; w = last xs ⟧ ⟹ ⦃λs. listp xs p s ∧ is_valid_list_C s x ⦄ all.list_insert_after' w x ⦃ λ_. listp (xs @ [x]) p⦄!"

7.5 list_find_last_node

7.5.1 list_find_last_node_inner_loop_content

definition list_find_last_node_inner_loop_content :: "list_C ptr ⇒ lifted_globals ⇒ list_C ptr option" **where**

```
  "list_find_last_node_inner_loop_content p ≡ DO oguard (λs. is_valid_list_C
s p);
        p <- ogets (λs. s[p]→next);
        oguard (λs. is_valid_list_C s p);
        oreturn p
      OD"
```

lemma list_find_last_node_inner_loop_content_correct : " ⟦ a ∈ set xs ; a ≠ NULL ⟧ ⟹ ovalidNF (λs. list xs pb s ∧ a_list a s ∧ s[a]→next ≠ NULL) (list_find_last_node_inner_loop_content a) (λr b. r ∈ set xs ∧ r ≠ NULL ∧ list xs pb b ∧ a_list r b)"

lemma list_find_last_node_inner_loop_content_pure : "ovalid P (list_find_last_node_inner_loop_conte a) (λ_. P)"

7.5.2 list_find_last_node_inner_loop

definition list_find_last_node_inner_loop :: "list_C ptr ⇒ lifted_globals ⇒ list_C ptr option" **where**

```
 "list_find_last_node_inner_loop p ≡ owhile (λp s. s[p]→next ≠ NULL) (λp.
DO oguard (λs. is_valid_list_C s p);
        p <- ogets (λs. s[p]→next);
        oguard (λs. is_valid_list_C s p);
        oreturn p
      OD) p"
```

lemma list_find_last_node_inner_loop_content_mesure : "a ≠ NULL ⟹ ovalid (λs. list xs pb s ∧ a_list a s ∧ s[a]→next ≠ NULL ∧ length (THE xs. list xs a s) = m) (list_find_last_node_inner_loop_content a) (λr s. length (THE xs. list xs r s) < m)"

lemma list_find_last_node_inner_loop_correct : "ovalidNF (list (xs @ [x]) pb) (list_find_last_node_inner_loop pb) (λr _. r = x)"

lemma list_find_last_node_inner_loop_pure : "ovalid P (list_find_last_node_inner_loop p) (λr. P)"

7.5.3 correct

lemma list_find_last_node_correct : "ovalidNF (listp (xs @ [x]) p) (all.list_find_last_node' p) (λr _. r = x) "

lemma list_find_last_node_correct2 : "xs = ys @ [w] $\Longrightarrow$ ovalidNF (listp xs p) (all.list_find_last_node' p) (λr _. r = w) "

lemma list_find_last_node_correct3 : "xs $\neq$ [] $\Longrightarrow$ ovalidNF (listp xs p) (all.list_find_last_node' p) (λr _. r = last xs)"

7.5.4 pure

lemma list_find_last_node_pure : "ovalid P (all.list_find_last_node' p) (λr. P)"

7.5.5 specialisations

lemma list_find_last_node_correct_ND : "xs $\neq$ [] $\Longrightarrow$ ⦃ listp xs l ⦄ gets_the (all.list_find_last_node' l) ⦃λxa s. xa = last xs⦄!"

lemma list_find_last_node_pure_ND : "⦃ P ⦄ gets_the (all.list_find_last_node' l) ⦃ λ_. P ⦄"

7.6 list_insert_back

7.6.1 correct

lemma list_insert_back_correct : "(x $\notin$ set xs $\wedge$ x $\neq$ NULL $\wedge$ x $\neq$ (ptr_coerce l)) $\Longrightarrow$ ⦃λs. listp xs l s $\wedge$ is_valid_list_C s x ⦄ all.list_insert_back' l x ⦃ λ _. listp (xs @ [x]) l ⦄!"

7.6.2 pure

lemma list_find_last_node_correct3' : "⦃λs. xs $\neq$ [] $\wedge$ listp xs l s⦄ gets_the (all.list_find_last_node' l) ⦃λx s. (xs $\neq$ [] $\wedge$ x = last xs)⦄"

lemma list_insert_back_pure: "⟦ ($\forall$s. P s $\longrightarrow$ P s[l := x]) ;
($\forall$s. P s $\longrightarrow$ P s[x$\rightarrow$next := s[(last xs)]$\rightarrow$next])
;
($\forall$s. P s $\longrightarrow$ P s[x$\rightarrow$next := NULL]) ;
($\forall$s. P s $\longrightarrow$ P s[(last xs)$\rightarrow$next := x]) ⟧

$\Longrightarrow$ ⦃ λs. listp xs l s $\wedge$ P s ⦄ all.list_insert_back' l x ⦃ λ_. P ⦄"

7.7 list_length

7.7.1 correct

definition `list_length_loop :: "(32 word × list_C ptr) ⇒ lifted_globals ⇒ (32 word × list_C ptr) option"` **where**
```
"list_length_loop ≡ owhile (λ(count, p) a. p ≠ NULL) (λ(count, p). DO y <-
oguard (λs. is_valid_list_C s p);
      p <- ogets (λs. s[p]→next);
      oreturn (count + 1, p)
   OD)"
```

lemma `list_length_loop_correct : "n ≤ 2 ^ LENGTH(32) - 1 ⟹ ovalidNF (list_length n p) (list_length_loop (0, p)) (λ(x,_) _. unat x = n)"`

lemma `list_length_correct : "n ≤ 2 ^ LENGTH(32) - 1 ⟹ ovalidNF (list_length_p n p) (all.list_length' p) (λr _. unat r = n)"`

7.8 list_pop

7.8.1 correct

lemma `list_pop_val : "listp (x # xs) p s  ⟹ listp xs p s[p := s[s[p]]→next]"`

lemma `list_pop_cons_correct : "⦃ listp (x # xs) p ⦄ all.list_pop' p ⦃ λr s. listp xs p s ∧ r = x ⦄!"`

lemma `list_pop_empty_correct : "⦃ listp [] p ⦄ all.list_pop' p ⦃ λr _. r = NULL ⦄!"`

7.8.2 pure

lemma `list_pop_pure : "(∀s. P s ⟶ P s[p := s[s[p]]→next]) ⟹ ⦃ P ⦄ all.list_pop' p ⦃ λ_. P ⦄"`

7.9 list_empty

7.9.1 correct

lemma `list_is_empty_correct : "ovalidNF (listp x p) (all.list_is_empty' p) (λr _. if x = [] then r = 1 else r = 0)"`

8 DEMO

Theorems proved during presentation

thm `fun_upd_def`
thm `list_is_empty`

lemma `list_empty_correct' : "⦃ λ s. is_valid_list_C'ptr s l ⦄ all.list_empty' l ⦃ λ_. listp [] l ⦄!"`

thm list_is_cons

lemma list_singleton_correct' : "⟦ x ≠ NULL ; x ≠ ptr_coerce l ⟧ ⟹
{| λs. is_valid_list_C s x ∧
is_valid_list_C'ptr s l |}
all.list_singleton_alt' l x
{| λ r. listp [x] l |}!"

thm list_st_update_ignore

lemma list_insert_front_correct' : "x ≠ NULL
⟹ {|λs. listp xs l s ∧ is_valid_list_C s x |} all.list_insert_front' l
x {|λr. listp (x#xs) l |}!"

end