Interactive Theorem Proving and Pointer Structures

Alfio Martini
alfio.martini@gmail.com

5th Workshop-School on Theoretical Computer Science
Passo Fundo - RS - Brasil

October 9-11, 2019

Contents

Outline

Main Objectives

Main Goals

- Introduce an Isabelle/HOL model for references and linked lists.
- Discuss how Hoare Logic can be used to reason about pointer structures.
- Present a proof methodology based on scripts and structured proofs.
- Apply these concepts to a non-trivial case study: deletion at the end of a linked list.

Outline

Essential Issues

Technical Difficulties

- Aliasing
- Local Reasoning
- Complexity of Proofs

Outline

Function Updates

$$f(a := v) \qquad \text{(FUNCTION UPDATE)}$$
$$(f(x := y))\ z = (if\ z = x\ then\ y\ else\ f\ z) \qquad \text{(SIMP RULE)}$$

$$\{i = j \wedge a[i] = 3\}\ a[i] := 4\ \{a[j] = 4\}$$

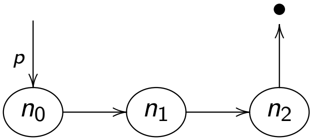
```
lemma "VARS (a::nat ⇒ int)
{i=j ∧ a(i) = 3}
a := a(i:=4)
{a(j) = 4}"
apply (vcg)
apply (simp)
done
```

References

$$\texttt{datatype}\ {}'\texttt{a}\ \texttt{ref}\ =\ \texttt{Null} \mid \texttt{Ref}\ {}'\texttt{a}$$

ADRESSES
$\texttt{addr} :: {}'\texttt{a}\ \texttt{ref} \Rightarrow {}'\texttt{a}$ $\texttt{addr}\ (\texttt{Ref}\ \texttt{x})\ =\ \texttt{x}$
LINKED LISTS $\texttt{next} :: {}'\texttt{a} \Rightarrow {}'\texttt{a}\ \texttt{ref}$
LOCAL HEAPS (FIELDS)
$\texttt{f} :: {}'\texttt{a} \rightarrow \texttt{values}$
HEAP UPDATE
$\texttt{f} := \texttt{f}((\texttt{addr r}) := \texttt{v})$

A Linked list for Students



node	*name*	*age*	*next*
*n*0	*Anne*	21	*n*1
*n*1	*Paul*	19	*n*2
*n*2	*July*	17	*Null*

Students Model in Isabelle

```
—‹ example:  0 ↦ 1 ↦ 2  ↦ Null ›
definition next_n::"nat ⇒nat ref" where
   "next_n≡(λn. Null)(0:=Ref 1,1:=Ref(2),2:=Null)"
definition name::"nat ⇒ string"  where
   "name ≡ (λn .'''')(0:=''Anne'',1:=''Paul'',2:=''July'')"
definition age::"nat ⇒ int"  where
   "age ≡ (λn . 0)(0:=19,1:=21,2:=17)"
definition p::"nat ref" where "p≡ Ref 0"
definition tmp::"nat ref" where "tmp≡p"

lemma "p^.name = ''Anne''" by (simp add:p_def name_def)
lemma "p^.next_n^.next_n=Ref 2" by (simp add:p_def next_n_def)
lemma "p^.next_n^.name = ''Paul''"
    by (simp add:p_def name_def next_n_def)
lemma "p^.next_n^.age = 21" by (simp add:p_def next_n_def age_def)
lemma "tmp^.age = 19" unfolding  p_def tmp_def age_def by simp
```

Notation

$$
\begin{array}{l}
\hline
f(r \rightarrow e) = f((addr\ r) := e) \\
r\hat{}.f := e = f := f(r \rightarrow e) \\
r\hat{}.f = f(addr\ r) \\
\hline
\end{array}
$$

Outline

Lists of Memory Addresses

$$\mathtt{List} :: ('a \Rightarrow' a\,\mathtt{ref}) \Rightarrow' a\,\mathtt{ref} \Rightarrow' a\,\mathtt{list} \Rightarrow \mathtt{bool}$$
$$\mathtt{List\ next\ r}\ [\,] = (\mathtt{r} = \mathtt{Null})$$
$$\mathtt{List\ next\ r}\ (\mathtt{a}\#\mathtt{as}) = (\mathtt{r} = \mathtt{Ref\ a} \wedge \mathtt{List\ next}\ (\mathtt{next\ a})\ \mathtt{as})$$

Essential Theorems

$$\mathtt{List\ next\ x\ as} \wedge \mathtt{List\ next\ x\ bs} \rightarrow \mathtt{as} = \mathtt{bs} \qquad \text{LFun}$$
$$\mathtt{List\ next\ x\ (as@bs)} \rightarrow \exists \mathtt{y}.\ \mathtt{List\ next\ y\ bs} \qquad \text{LRef}$$
$$\mathtt{List\ next(\ next\ a)\ x\ as} \rightarrow \mathtt{a} \notin \mathtt{set\ as} \qquad \text{LAci}$$
$$\mathtt{List\ next\ x\ as} \rightarrow \mathtt{distinct\ as} \qquad \text{LDist}$$
$$\mathtt{a} \notin \mathtt{set\ as} \quad \mathtt{List\ (next(a := y))\ x\ as} \rightarrow \mathtt{List\ next\ x\ as} \qquad \text{LSep}$$

Outline

Hoare Tiples (Partial Correctness Assertions)

$$\{P\}\ c\ \{Q\}$$

IF THE PROGRAM c STARTS EXECUTING IN A STATE WHERE THE ASSERTION P IS TRUE, THEN IF c TERMINATES, IT DOES SO IN A STATE WHERE THE ASSERTION Q HOLDS.

An Introductory Example

```
lemma "VARS (next::'a ⇒ 'a ref) (p::'a ref)
          (ps::'a list) (k::nat) j
{List next p Ps ∧ j = length Ps}
k:=0;
WHILE p ≠ Null
INV {∃ as. List next p as ∧ length as + k = j}
DO p := p^.next; k := k+1 OD
{j = k ∧ List next p []}"
apply (vcg)
  apply (fastforce)
  apply (fastforce)
  apply (fastforce)
done
```

Verification Conditions

```
proof (prove)
goal (3 subgoals):
 1. ⋀next p ps k j.
       List next p Ps ∧ j = length Ps ⟹
       ∃as. List next p as ∧ length as + 0 = j
 2. ⋀next p ps k j.
       (∃as. List next p as ∧ length as + k = j) ∧
       p ≠ Null ⟹
       ∃as. List next (next (addr p)) as ∧
            length as + (k + 1) = j
 3. ⋀next p ps k j.
       (∃as. List next p as ∧ length as + k = j) ∧
       ¬ p ≠ Null ⟹
       j = k ∧ List next p []
```

Structured Proof - Second VC

```
proof (vcg)
  fix "next" and p::"'a ref" and  ps k j
  assume ass: "(∃as. List next p as ∧
               length as + k = j) ∧  p ≠ Null"
  show " ∃as. List next (next (addr p)) as ∧
              length as + (k + 1) = j"
    proof -
       from ass  obtain as  where
       list: "List next p as " and len:"length as + k = j"
        and nNull: "p ≠ Null" by blast
       from nNull and list  obtain a bs
         where   "p = Ref a" and  "as = a # bs"
          and "List next (next a) bs"  by auto
       from this and len  have "length bs + k + 1 = j"
        and "List next (next (addr p)) bs" by auto
      from this show ?thesis by auto
    qed
qed (fastforce)+
```

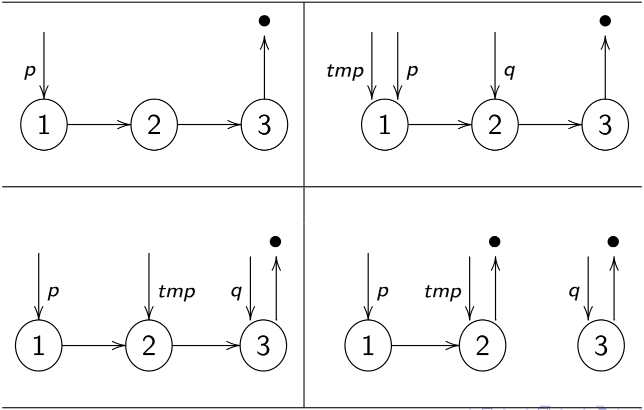
Outline

Hoare Triple - Deletion at the End

```
lemma "VARS p next q tmp  ps qs
{List next p Ps  ∧ p ≠ Null}
IF p^.next = Null
  — ‹Ps has exactly one element›
  THEN p:= Null;ps:=[];qs:=Ps
ELSE
  — ‹Ps has at least two elements ›
  tmp := p; q:= p^.next;
  ps :=[hd Ps]; qs := tl Ps;
  WHILE q^.next ≠ Null
  INV  {inv_del_end}
  DO
     tmp := q; ps := ps @ [hd qs];
     q:= q^.next;qs := tl qs
  OD;
  tmp^.next := Null
FI
{∃ a as. List next p as ∧ as @ [a] = Ps}"
```

Invariant Intuitively

Deletion at the end - Invariant Assertion



Invariant in Isabelle

```
INV  {p ≠ Null ∧ List next p Ps
      ∧ List next q qs
      ∧ List (next(last ps := Null)) p ps
      ∧ ps @ qs = Ps
      ∧ set ps ∩ set qs = {}
      ∧ next (last ps) = q
      ∧ last ps = addr tmp
      ∧ ps ≠ [] ∧ qs ≠ []
     }
```

Verification Conditions

ISABELLE TOOL

Proof Script

ISABELLE TOOL

Isar Proof - Third VC

```
fix p "next" q tmp ps qs
assume ass:"(∃y. p = Ref y) ∧ List next p Ps ∧
  List next q qs ∧ List (next(last ps := Null)) p ps
  ∧ ps @ qs = Ps ∧ set ps ∩ set qs = {}
  ∧ next (last ps) = q ∧ last ps = addr tmp
  ∧ ps ≠ [] ∧ qs ≠ [] ∧ next (addr q) = Null"
show "∃a as.
        List (next(tmp → Null)) p as ∧ as @ [a] = Ps"
 proof -
   from ass have lqs:"List next q qs" and "qs ≠ []"
     and nq:"next (addr q) = Null" and
    lps: "List (next(last ps := Null)) p ps"
   and pq: "ps @ qs = Ps" and
       tmp:"last ps = addr tmp" by auto
   from this(1-2) obtain a as where "q=Ref a" and
     qs:"qs = a # as" and "List next (next a) as"
     by (induction qs) simp_all
   from this(3) and nq and ‹q=Ref a›
       have "as = []" by simp
   from pq and this and qs have "ps @ [a] = Ps" by simp
   from lps and this and tmp  show ?thesis
     by auto
 qed
```

Outline

Concluding Remarks

- A detailed presentation of techniques to reason about pointer structures using standard Hoare Logic in Isabelle/HOL.
- Linked chunks of memory are represented as total functions from addresses to references.
- These linked structures are abstracted to a (finite) list of addresses, so that standard methods of reasoning can be applied.
- Proofs and invariants tend to be complex.
- Proof exploration with proof scripts.
- Proof drafts and proof documentation with Isar e Isabelle automatic proof methods.