Overview of the seplogC Library (Implementation Notes)

Reynald Affeldt

What is this Document?

This document is an overview of the seplogC library for formal verification using the Coq proof-assistant of programs written in a subset of the C language. The purpose of this overview is essentially to list up the numerous lemmas and tactics that are useful to formally verify programs in practice. Writing is in progress.

The seplogC library has been used to formally (fix and) verify a parsing function from PolarSSL, an implementation of the TLS protocol. Technical details about the encoding in Coq of C and its application to PolarSSL can be found in [2]. The Coq documentation for the seplogC library can be found online [3]. The verification of the mandatory reverse-list example from Sect. 1.3 can also be found online [3].

Contents

1 A Subset of the C Language **2**
- 1.1 Values 2
- 1.2 Expressions 2
- 1.3 Commands 3

2 Assertions **3**

3 Generic Proof Strategy **3**

4 Naming Convention for Lemmas and Tactics **5**

5 Lemmas **5**
- 5.1 Manipulation of Assertions 5
- 5.2 Lemmas about Entailment 6
  - 5.2.1 Structural Rules 6
  - 5.2.2 Logical Rules 6
  - 5.2.3 Logical Rules with Pure Assertions 6
  - 5.2.4 Turn Substitutions into Substitutions in the Store 7
- 5.3 Lemmas about Hoare Triples 7
  - 5.3.1 Structural Rules 7
  - 5.3.2 Logical Rules 7
  - 5.3.3 Inference Rules 8

6 Tactics **9**
- 6.1 Manipulation of Assertions 9
- 6.2 Tactics about Entailment 9
  - 6.2.1 Structural Rules 9
  - 6.2.2 Logical Rules 9
  - 6.2.3 Logical Rules with Pure Assertions 10
  - 6.2.4 Rewriting and Substitution 10

6.3 Tactics About Hoare Triples 11
6.3.1 Enhanced Inference Rules 11
6.3.2 Structural Rules 11
6.3.3 Logical Rules 11
6.3.4 Logical Rules with Pure Assertions 12

1 A Subset of the C Language

1.1 Values

`` `( x )_ n `` is notation for a finite-size (`n` bits) unsigned integer with decimal value `x` (if it exists). `` `( x )s_ n `` is for signed integers. A *physical value* of type t is a sequence of bytes l such that $|l| =$ `sizeof`(t). In particular, a physical value that represents a C structure is a sequence of bytes including padding. `[ i ]p` where `i` is a finite-size integer is notation for physical values (type inferred from the context). The physical values with null bytes are noted $pv0$. A *logical value* is a slightly more abstract view of physical values. For example, a logical value that represents a C structure is a sequence of basic logical values without padding (but still respecting alignment restrictions). Physical values are ranged over by pv and logical values by lv. When a physical value and a logical value correspond to the same sequence of bytes, we write pv Ⓒ lv.

1.2 Expressions

s ranges over stores of variables. str ranges over strings. e ranges over expressions of the C language. See Table 1 for C expressions. $\mathcal{E}[\![e]\!]_s$ is the physical value of e when evaluated in the store s. Expressions are side-effect free: their evaluation never accesses the heap. An expression that evaluates to the null physical value is interpreted as false, and true otherwise.

<table>
<tr><th colspan="2">Numerical operators</th></tr>
<tr><td>e_1 \+ e_2</td><td>arithmetic or pointer addition, depending on the type</td></tr>
<tr><td>e_1 \- e_2</td><td>arithmetic subtraction</td></tr>
<tr><td>e_1 * e_2</td><td>arithmetic multiplication</td></tr>
<tr><td>e_1 \& e_2</td><td>bitwise and</td></tr>
<tr><td>e_1 \| e_2</td><td>bitwise or</td></tr>
<tr><td>e_1 \<< e_2</td><td>left-shift</td></tr>
<tr><td>e \% n</td><td>modulo 2^n</td></tr>
<tr><th colspan="2">Relational operators (returns 0 or 1, as an unsigned int)</th></tr>
<tr><td>e_1 \= e_2</td><td>equality between arithmetic types or pointers, depending on the type</td></tr>
<tr><td>e_1 \!= e_2</td><td>inequality test</td></tr>
<tr><td>e_1 { \<, \<=, \>, >= } e_2</td><td>arithmetic comparisons</td></tr>
<tr><td>e_1 \&& e_2</td><td>logical and</td></tr>
<tr><td>e_1 \|| e_2</td><td>logical or</td></tr>
<tr><th colspan="2">Others</th></tr>
<tr><td>e &-> f</td><td>returns the address of the field f when e points to a structure
of the appropriate type</td></tr>
<tr><td>e \? f \: g</td><td>if-then-else</td></tr>
<tr><td>[; t ;] e</td><td>safe cast (no data loss nor misinterpretation)</td></tr>
<tr><td>(int) e</td><td>e.g., safe cast to (signed) int</td></tr>
<tr><td>{; t ;} e</td><td>unsafe cast</td></tr>
<tr><td>(UINT) e</td><td>e.g., unsafe cast to unsigned int</td></tr>
<tr><td>NULL</td><td>pointers of value 0</td></tr>
</table>

Table 1: C expressions

b ranges over boolean expressions. Boolean expressions are of the form $({}^b\ e)$ or ${}^b\neg b$. $\mathcal{B}[\![b]\!]_s$ is the boolean

skip	does nothing
str `<-` e	variable assignment
str `<-*` e	assignment of str with the contents of address e
e_1 `*<-` e_2	update the contents of the address e_1 with e_2
c_1 `;` c_2	sequence
`If` b `Then` c_1 `Else` c_2	conditional branch
`While` e `{{`c `}}`	while-loop
`For`$(c_1 ; c_2, e ; c_3)$ `{{`c `}}`	c_1 `;` c_2 `;` `While` $(^b\, e)$ `{{`c `;` c_3 `;` c_2 `}}`
str `+<-` e	str `<-` $str + e$
str`++`	str `+<-` 1

Table 2: C commands

evaluation of b in the store s. When a boolean expression has no free variable, it is ground and its physical value (independent of the store) can be noted $[b]_{gb}$.

$e\{e'/str\}$ is the expression e where the occurrences of the variable str (of type, say, t') have been replaced with e' (of compatible type t').

1.3 Commands

c ranges over commands. See Table 2 for C commands.

Example Swap two cells in memory:

```
Definition swap :=
  "z" <-* %"x" ;
  "v" <-* %"y" ;
  %"x" *<- %"v" ;
  %"y" *<- %"z".
```

In-place reverse-list:

```
Definition reverse_list :=
  _ret <- NULL ;
  While (\~b \b __i \= NULL) {{
    _rem <-* __i &-> _next;
    __i &-> _next *<- __ret ;
    _ret <- __i ;
    _i <- __rem
  }}.
```

2 Assertions

h ranges over heaps. P, Q, R, Inv range over assertions of Coq type `store -> heap -> Prop` (so-called shallow embedding). See Table 3 for a (non-exhaustive) set of assertions. boolean predicates are captured by boolean expressions b. $\mathcal{P}$, etc. have type `Prop`. The precise definitions of the substitutions can be found in Table 4.

3 Generic Proof Strategy

The original goal is a Hoare triple $\{\ P\ \}\ c\ \{\ Q\ \}$. The general strategy is to break down this triple according to the syntax of c (using the inference rules for the composition of commands) and then according to the shape of P and Q (using the frame rule). When the subgoals are minimal triples $\{\ P_1\ \}\ c_1\ \{\ Q_1\ \}$ where c_1

Logical Connectives

F	never holds
T	holds for any heap
ϵ	holds for the empty heap
$\star$	separating conjunction
$\vee$, $\wedge$	lifting of the Coq $\vee$ and $\wedge$
$(\texttt{sepex}\ i\texttt{,}P\ i)$	holds when there exists an i such that $P\ i$ holds
$a \mapsto_{\text{lV}} v$	maps an address (physical value) a to a logical value v
$e \mapsto_{\text{le}} v$	maps an address (expression) e to a logical value v
$e \Mapsto l$	holds when e points to the list of physical values l
$e \overset{\text{fit}}{\Mapsto} l$	holds when $e \Mapsto l$ holds and $\texttt{!}\ (\mathcal{E}[\![e]\!]_s + \texttt{sizeof}(t) \times \lvert l\rvert < 2^{32})$ also

Pure Assertions

$\texttt{!}\ P$	holds when P holds for the store and the heap is empty (P has type $\texttt{store -> Prop}$)
$\texttt{`!}\ b$	holds when $\texttt{!}\ \mathcal{B}[\![b]\!]$ holds
$\texttt{!!}\ \mathcal{P}$	holds when $\mathcal{P}$ holds and the heap is empty

Substitutions

$P\texttt{\{}str\ \texttt{<-}\ e\texttt{\}}$	holds when the occurrences of str in P are replaced with e
$P\texttt{\{}str\ \texttt{<-}\ *e\texttt{\}}$	holds when str is replaced with $*e$ in P
$P\texttt{\{}str\ \texttt{<-}\ e\ \texttt{\&->}\ f\texttt{\}}$	holds when str is replaced with $e\ \texttt{\&->}\ f$ in P
$P\texttt{\{}str\ \texttt{<-}\ l\texttt{\{}ei, e, i\texttt{\}}\texttt{\}}$	holds when str is replaced with the ith element of array l in P
$P^{\text{fit}}\texttt{\{}str\ \texttt{<-}\ l\texttt{\{}ei, e, i\texttt{\}}\texttt{\}}$	holds when str is replaced with the ith element of array l in P (with the additional constraint that the array fits in memory)

Table 3: Assertions (see Table 4 for the definitions of the substitutions)

$P\texttt{\{}x\ \texttt{<-}\ *e\texttt{\}}$
$\quad \overset{\text{def}}{=} \exists pv, s, h. h(\mathcal{E}[\![e]\!]_s) = pv \wedge P\, s\{pv/str\}\, h$
$P\texttt{\{}str\ \texttt{<-}\ e\ \texttt{\&->}\ f\texttt{\}}$
$\quad \overset{\text{def}}{=} \exists lvs, pv, lv, s, h. lvs_f = lv \wedge pv \copyright lv \wedge (e \mapsto_{\text{lV}} lvs \star \mathsf{T})s\, h \wedge (P\{pv/str\})s\, h$
$P\texttt{\{}str\ \texttt{<-}\ l\texttt{\{}ei, e, i\texttt{\}}\texttt{\}}$
$\quad \overset{\text{def}}{=} \exists s, h. i < \lvert l\rvert \wedge ((\texttt{`!}\ (^b\ e_i = e + i)) \star (e \Mapsto l) \star \mathsf{T})s\, h \wedge (P\texttt{\{}str\ \texttt{<-}\ l_i\texttt{\}})s\, h$
$P^{\text{fit}}\texttt{\{}str\ \texttt{<-}\ l\texttt{\{}ei, e, i\texttt{\}}\texttt{\}}$
$\quad \overset{\text{def}}{=} \exists s, h. i < \lvert l\rvert \wedge ((\texttt{`!}\ (^b\ e_i = e + i)) \star (e \overset{\text{fit}}{\Mapsto} l) \star \mathsf{T})s\, h \wedge (P\texttt{\{}str\ \texttt{<-}\ l_i\texttt{\}})s\, h$

Table 4: Definition of the substitutions of Table 3

is some basic command, one looks for appropriate lemmas or tactics to turn the subgoal into an entailment of the form P' `===>` Q' (i.e., if P' holds then Q' holds). The latter entailment usually features substitutions; we use lemmas to turn the various kinds of substitutions into substitutions w.r.t. the store only. Finally, the resulting entailment is further broken down according to the shape of P' and Q' to finally reduce the entailment to a Coq logical formula, for which lemmas from the standard library or the user libraries can be used.

Example

$$\{\ "x" \mapsto_{\mathrm{le}} a \star "y" \mapsto_{\mathrm{le}} b\ \}\ swap\ \{\ "x" \mapsto_{\mathrm{le}} b \star "y" \mapsto_{\mathrm{le}} a\ \}$$

↓ tactics `Hoare_seq_ext` `` `! `` $z = a$ `Hoare_Frame_idx (0 :: nil)(0 :: 1 ::nil)` → other subgoals

$$\{\ "x" \mapsto_{\mathrm{le}} a\ \}\ z\ \texttt{<-*}\ "x"\ \{\ (\texttt{`!}\ z = a) \star "x" \mapsto_{\mathrm{le}} a\ \}$$

↓ lemma `hoare_lookup_stren`

$$"x" \mapsto_{\mathrm{le}} a\ \texttt{===>}\ \texttt{`!}\ z = a \star "x" \mapsto_{\mathrm{le}} a\{z\ \texttt{<-}\ \ast "x"\}$$

↓ lemma `ent_R_lookup_trans`

$$"x" \mapsto_{\mathrm{le}} a\ \texttt{===>}\ (\texttt{`!}\ z = a \star "x" \mapsto_{\mathrm{le}} a)\{z\ \texttt{<-}\ a\}$$

↓ tactics `Ent_R_subst_con_distr` and `Ent_R_subst_apply`

first subgoal proved

4 Naming Convention for Lemmas and Tactics

`con*`	lemma about the separating conjunction $\star$ of the form $\cdots$ `<==>` $\cdots$
`ent_L_*`	lemma with conclusion $\cdot$ `===>` $\cdot$ acting on the left
`ent_R_*`	lemma with conclusion $\cdot$ `===>` $\cdot$ acting on the right
`hoare_L_*`	lemma with conclusion $\{\ \cdot\ \}\ \cdot\ \{\ \cdot\ \}$ acting on the precondition
`hoare_R_*`	lemma with conclusion $\{\ \cdot\ \}\ \cdot\ \{\ \cdot\ \}$ acting on the postcondition
`Ent_*`	tactic for goals $\cdot$ `===>` $\cdot$
`Hoare_*`	tactic for goals $\{\ \cdot\ \}\ \cdot\ \{\ \cdot\ \}$

5 Lemmas

5.1 Manipulation of Assertions

$$P \star (Q \star R)\ \overset{\texttt{conA}}{\texttt{<==>}}\ (P \star Q) \star R \qquad P \star Q \star R\ \overset{\texttt{conCA}}{\texttt{<==>}}\ Q \star P \star R$$

$$\epsilon \star P\ \overset{\texttt{coneP}}{\texttt{<==>}}\ P \qquad \mathsf{F} \star P\ \overset{\texttt{conFP}}{\texttt{<==>}}\ \mathsf{F}$$

$$R \star (P \vee Q)\ \overset{\texttt{conDr}}{\texttt{<==>}}\ (R \star P) \vee (R \star Q) \qquad (P \vee Q) \star R\ \overset{\texttt{conDl}}{\texttt{<==>}}\ (P \star R) \vee (Q \star R)$$

$$\texttt{`!}\ b\ \overset{\texttt{bbang\_dup}}{\texttt{<==>}}\ \texttt{`!}\ b \star \texttt{`!}\ b.$$

`wp_assign_or`

$$(P \vee Q)\{str\ \texttt{<-}\ e\}\ \texttt{<==>}\ (P\{str\ \texttt{<-}\ e\}) \vee (Q\{str\ \texttt{<-}\ e\})$$

`wp_assign_ex`

$$(\texttt{sepex}\ i\,,P i)\{str\ \texttt{<-}\ e\}\ \texttt{<==>}\ (\texttt{sepex}\ i\,,(P\,i)\{str\ \texttt{<-}\ e\})$$

`wp_lookup_fldp_ex`

$$(\texttt{sepex}\ x\,,P\,x)\{str\ \texttt{<-}\ e\ \texttt{\&->}\ f\}\ \texttt{<==>}\ (\texttt{sepex}\ x\,,(P\,x)\{str\ \texttt{<-}\ e\ \texttt{\&->}\ f\})$$

5.2 Lemmas about Entailment

5.2.1 Structural Rules

`monotony` $\quad\dfrac{P \texttt{ ===> } Q \qquad R \texttt{ ===> } S}{P \star R \texttt{ ===> } Q \star S}$

`monotony_L` $\quad\dfrac{P \texttt{ ===> } Q}{R \star P \texttt{ ===> } R \star Q}$

`monotony_R` $\quad\dfrac{P \texttt{ ===> } Q}{P \star R \texttt{ ===> } Q \star R}$

NB: Generalized by the tactic `Ent_monotony` (see Sect. 6.2).

5.2.2 Logical Rules

`ent_L_F` $\quad\dfrac{}{\mathsf{F} \texttt{ ===> } Q}$

`ent_R_T` $\quad\dfrac{}{P \texttt{ ===> } \mathsf{T}}$

`ent_id` $\quad\dfrac{}{P \texttt{ ===> } P}$

`ent_trans` $\quad\dfrac{P \texttt{ ===> } Q \qquad Q \texttt{ ===> } R}{P \texttt{ ===> } R}$

`ent_R_or_1` $\quad\dfrac{P \texttt{ ===> } R}{P \texttt{ ===> } R \vee Q}$

`ent_R_or_2` $\quad\dfrac{P \texttt{ ===> } Q}{P \texttt{ ===> } R \vee Q}$

NB: The tactic `Ent_R_or_1` generalizes `ent_or_R_1` (see Sect. 6.2).

5.2.3 Logical Rules with Pure Assertions

`ent_bang_contract` $\quad\dfrac{}{\texttt{!}\ P \texttt{ ===> } \epsilon}$

Variants: `ent_bbang_contract` $\quad\dfrac{}{\texttt{`!}\ b \texttt{ ===> } \epsilon}$

`ent_L_bbang` $\quad\dfrac{\epsilon \texttt{ ===> } R}{\texttt{`!}\ b \texttt{ ===> } R}$

`ent_L_con_bbang` $\quad\dfrac{P \texttt{ ===> } R}{P \star \texttt{`!}\ b \texttt{ ===> } R}$

`ent_L_bbang_con` $\quad\dfrac{P \texttt{ ===> } R}{\texttt{`!}\ b \star P \texttt{ ===> } R}$

NB: Generalized by `Ent_L_contract_bbang` (see Sect. 6.2)

`ent_R_sbang`

$$\frac{}{\epsilon \texttt{ ===> } \texttt{!!}\ P}$$

`ent_L_sbang_con`

$$\frac{P \to Q \texttt{ ===> } R}{\texttt{!!}\ \mathcal{P} \star \mathcal{Q} \texttt{ ===> } \mathcal{R}}$$

`ent_sbang_sbang`

$$\frac{\mathcal{P} \to \mathcal{Q}}{\texttt{!!}\ \mathcal{P} \texttt{ ===> } \texttt{!!}\ \mathcal{Q}}$$

`ent_R_sbang` $\quad\dfrac{\mathcal{P}}{\epsilon \texttt{ ===> } \texttt{!!}\ \mathcal{P}}$

Variant: `ent_R_sbang_con` $\quad\dfrac{\mathcal{R} \qquad P \texttt{ ===> } Q}{P \texttt{ ===> } \texttt{!!}\ \mathcal{R} \star Q}$

5.2.4 Turn Substitutions into Substitutions in the Store

`ent_R_lookup_trans`

$$\frac{pv \copyright lv \qquad R \texttt{ ===> } e \mapsto_{\mathrm{le}} lv \star \mathsf{T} \qquad R \texttt{ ===> } P\{str \texttt{ <- } [pv]_c\}}{R \texttt{ ===> } P\{str \texttt{ <- } *e\}}$$

`ent_R_lookup_fldp`

$$\frac{pv \copyright lvs_f \qquad e \mapsto_{\mathrm{lV}} lvs \texttt{ ===> } P\{str \texttt{ <- } [pv]_c\}}{e \mapsto_{\mathrm{lV}} lvs \texttt{ ===> } P\{str \texttt{ <- } e \texttt{ \&-> } f\}}$$

`ent_R_lookup_fldp_trans`

$$\frac{pv \copyright lvs_f \qquad R \texttt{ ===> } e \mapsto_{\mathrm{lV}} lvs \star \mathsf{T} \qquad R \texttt{ ===> } P\{str \texttt{ <- } [pv]_c\}}{R \texttt{ ===> } P\{str \texttt{ <- } e \texttt{ \&-> } f\}}$$

`ent_R_lookup_mapstos_trans`

$$\frac{i < |l| \qquad R \texttt{ ===> } \texttt{`!} \ ({}^b\ e_i = e + i) \star (e \Mapsto l) \star \mathsf{T} \qquad R \texttt{ ===> } P\{str \texttt{ <- } l_i\}}{R \texttt{ ===> } P\{str \texttt{ <- } l\{e_i, e, i\}\}}$$

`ent_R_lookup_mapstos_fit_trans`

$$\frac{i < |l| \qquad R \texttt{ ===> } \texttt{`!} \ ({}^b\ e_i = e + i) \star (e \Mapsto l) \star \mathsf{T} \qquad R \texttt{ ===> } P\{str \texttt{ <- } l_i\}}{R \texttt{ ===> } P^{\mathrm{fit}}\{str \texttt{ <- } l\{e_i, e, i\}\}}$$

5.3 Lemmas about Hoare Triples

5.3.1 Structural Rules

`hoare_stren`

$$\frac{P \texttt{ ===> } P' \qquad \{\ P'\ \}\ c\ \{\ Q\ \}}{\{\ P\ \}\ c\ \{\ Q\ \}}$$

`hoare_weak`

$$\frac{\{\ P\ \}\ c\ \{\ Q'\ \} \qquad Q \texttt{ ===> } Q'}{\{\ P\ \}\ c\ \{\ Q\ \}}$$

5.3.2 Logical Rules

`hoare_L_F`

$$\frac{}{\{\ \mathsf{F}\ \}\ c\ \{\ Q\ \}}$$

`hoare_LR_and`

$$\frac{\{\ P\ \}\ c\ \{\ Q\ \} \qquad \{\ P'\ \}\ c\ \{\ Q'\ \}}{\{\ P \wedge P'\ \}\ c\ \{\ Q \wedge Q'\ \}}$$

`hoare_L_or`

$$\frac{\{\ P_1\ \}\ c\ \{\ Q\ \} \qquad \{\ P_2\ \}\ c\ \{\ Q\ \}}{\{\ P_1 \vee P_2\ \}\ c\ \{\ Q\ \}}$$

`hoare_R_or_1`

$$\frac{\{\ P\ \}\ c\ \{\ Q_1\ \}}{\{\ P\ \}\ c\ \{\ Q_1 \vee Q_2\ \}}$$

`hoare_R_or_2`

$$\frac{\{\ P\ \}\ c\ \{\ Q_2\ \}}{\{\ P\ \}\ c\ \{\ Q_1 \vee Q_2\ \}}$$

NB: The tactic `Hoare_L_or` (see Sect. 6.3) generalizes `hoare_L_or`.

`hoare_R_ex`

$$\frac{\exists x, \{\ P\ \}\ c\ \{\ Q\,x\ \}}{\{\ P\ \}\ c\ \{\ \exists x, Q\,x\ \}}$$

`hoare_L_ex`

$$\frac{\forall x, \{\ P\,x\ \}\ c\ \{\ Q\ \}}{\{\ \exists x, P\,x\ \}\ c\ \{\ Q\ \}}$$

NB: The tactic `Hoare_L_ex` (see Sect. 6.3) generalizes `hoare_L_ex`.

5.3.3 Inference Rules

Basic Commands

`hoare0_assign`

$$\frac{}{\{\, P\{str \texttt{ <- } e\} \,\}\ str \texttt{ <- } e \ \{\, P \,\}}$$

Variant: `hoare_assign_stren`

$$\frac{P \texttt{ ===> } Q\{str \texttt{ <- } v\}}{\{\, P \,\}\ str \texttt{ <- } v \ \{\, Q \,\}}$$

`hoare0_lookup`

$$\frac{}{\{\, P\{x \texttt{ <- } *e\} \,\}\ x \texttt{ <-* } e \ \{\, P \,\}}$$

`hoare_lookup_stren`

$$\frac{P \texttt{ ===> } Q\{str \texttt{ <- } *e\}}{\{\, P \,\}\ x \texttt{ <-* } e \ \{\, Q \,\}}$$

Composition of Commands

`hoare_seq`

$$\frac{\{\, P \,\}\ c_1 \ \{\, R \,\} \qquad \{\, R \,\}\ c_2 \ \{\, Q \,\}}{\{\, P \,\}\ c_1 \texttt{ ; } c_2 \ \{\, Q \,\}}$$

Variant: `hoare_assign_seq_stren`

$$\frac{P \texttt{ ===> } R\{str \texttt{ <- } e\} \qquad \{\, R \,\}\ c \ \{\, Q \,\}}{\{\, P \,\}\ str \texttt{ <- } e; c \ \{\, Q \,\}}$$

`hoare_ifte_bang`

$$\frac{\{\, P \star \texttt{`! } b \,\}\ c_1 \ \{\, Q \,\} \qquad \{\, P \star \texttt{`! } {}^b\neg b \,\}\ c_2 \ \{\, Q \,\}}{\{\, P \,\}\ \texttt{If } b \texttt{ Then } c_1 \texttt{ Else } c_2 \ \{\, Q \,\}}$$

`hoare_forloop2`

$$\frac{\{\, P \,\}\ c_1 \texttt{ ; } c_2 \ \{\, Inv \,\} \qquad (\texttt{`! } {}^b\neg({}^b\, e)) \star Inv \texttt{ ===> } Q \qquad \{\, \texttt{`! } ({}^b\, e) \star Inv \,\}\ c \texttt{ ; } c_3 \texttt{ ; } c_2 \ \{\, Q \,\}}{\{\, P \,\}\ \texttt{For}(c_1; c_2, e; c_3)\texttt{\{}\{c\}\texttt{\}} \ \{\, Q \,\}}$$

Variants of Basic Commands

`hoare_mutation_local_forward_ground_le`

$$\frac{e_2 \text{ has no free variable} \qquad e_2 \ \text{ⓒ}\ lv'}{\{\, e_1 \mapsto_{\text{le}} lv \,\}\ e_1 \texttt{ *<- } e_2 \ \{\, e_1 \mapsto_{\text{le}} lv' \,\}}$$

Array Access

`hoare_lookup_mapstos`

$$\frac{\begin{array}{c} t \text{ is the type of } str \quad l \text{ is a list of physical values of type t} \\ e_i, e \text{ are expressions of type } *t \quad |l| \times \texttt{sizeof}(t) < 2^{31} \end{array}}{\{\, P\{str \texttt{ <- } l\{e_i, e, i\}\} \,\}\ str \texttt{ <-* } e_i \ \{\, P \,\}}$$

Variant: `hoare_lookup_mapstos_stren`

`hoare_lookup_mapstos_fit`

$$\frac{\begin{array}{c} t \text{ is the type of } str \quad l \text{ is a list of physical values of type t} \\ e_i, e \text{ are expressions of type } *t \quad |l| \times \texttt{sizeof}(t) < 2^{31} \end{array}}{\{\, P^{\text{fit}}\{str \texttt{ <- } l\{e_i, e, i\}\} \,\}\ str \texttt{ <-* } e_i \ \{\, P \,\}}$$

Variant: `hoare_lookup_mapstos_fit_stren`

Structure Access

`hoare_lookup_fldp`

$$\frac{e \text{ is a pointer to a structure}}{\{\, P\{str\ \texttt{<-}\ e\ \texttt{\&->}\ f\}\,\}\ str\ \texttt{<-*}\ e\ \texttt{\&->}\ f\ \{\, P \,\}}$$

Variant: `hoare_lookup_fldp_stren`

$$\frac{P\ \texttt{===>}\ Q\{str\ \texttt{<-}\ e\ \texttt{\&->}\ f\}}{\{\, P \,\}\ str\ \texttt{<-*}\ e\ \texttt{\&->}\ f\ \{\, Q \,\}}$$

`hoare_mutation_fldp_local_forward_ground_lV`

$$\frac{pv \text{ is a physical value} \qquad \text{e has no free variable} \qquad e \copyright lv}{\{\, pv \mapsto_{\text{lV}} lvs \,\}\ pv\ \texttt{\&->}\ f\ \texttt{*<-}\ e\ \{\, pv \mapsto_{\text{lV}} lvs\{lv/f\} \,\}}$$

`hoare_mutation_fldp_local_forward_ground_le`

$$\frac{e_2 \text{ has no free variable} \qquad e_2 \copyright lv}{\{\, e_1 \mapsto_{\text{le}} lvs \,\}\ e_1\ \texttt{\&->}\ f\ \texttt{*<-}\ e_2\ \{\, e_1 \mapsto_{\text{le}} lvs\{lv/f\} \,\}}$$

`hoare_mutation_fldp_subst_ityp/ptr`

$$\frac{P\ \texttt{===>}\ str = e \qquad \{\, P \,\}\ e_1\{e/str\}\ \texttt{\&->}\ f\ \texttt{*<-}\ e_2\{e/str\}\ \{\, Q \,\}}{\{\, P \,\}\ e_1\ \texttt{\&->}\ f\ \texttt{*<-}\ e_2\ \{\, Q \,\}}$$

6 Tactics

6.1 Manipulation of Assertions

`Bbang2sbang` turns all the occurrences of $\texttt{`!}\ b$ into $\texttt{!!}\ [b]_{gb}$ when there is no free variable in b.

6.2 Tactics about Entailment

6.2.1 Structural Rules

`Ent_L_stren_by P l`

$$\frac{l \subseteq R \qquad l\ \texttt{===>}\ P \qquad P \star R\ \texttt{===>}\ Q}{R\ \texttt{===>}\ Q}$$

`Ent_permut`

$$\frac{\sigma \text{ permutation}}{\bigstar_i P_i\ \texttt{===>}\ \bigstar_{\sigma(i)} P_i}$$

`Ent_decompose l1 l2`

$$\frac{\bigstar_{i \in l_1} P_i\ \texttt{===>}\ \bigstar_{j \in l_2} Q_j \qquad \bigstar_{i \notin l_1} P_i\ \texttt{===>}\ \bigstar_{j \notin l_2} Q_j}{\bigstar_i P_i\ \texttt{===>}\ \bigstar_j Q_j}$$

`Ent_monotony` simplifies $P\ \texttt{===>}\ Q$ by eliminating assertions that appear in both P and Q. `Ent_monotony0` is a faster but simpler version of `Ent_monotony`, e.g., it solves $P\ \texttt{===>}\ P \star (\texttt{`!}\ b = b)$.

6.2.2 Logical Rules

Here, `n` is the occurrence number of the logical connective ($\vee$, $\exists$).

`Ent_L_or n`

$$\frac{\cdots P \cdots\ \texttt{===>}\ Q \qquad \cdots R \cdots\ \texttt{===>}\ Q}{\cdots P \vee R \cdots\ \texttt{===>}\ Q}$$

`Ent_R_or_1 n`

$$\frac{P\ \texttt{===>}\ \cdots Q \cdots}{P\ \texttt{===>}\ \cdots Q \vee R \cdots}$$

`Ent_R_or_2 n`

$$\frac{P\ \texttt{===>}\ \cdots R \cdots}{P\ \texttt{===>}\ \cdots Q \vee R \cdots}$$

`Ent_L_ex n k`

$$\frac{\forall k, \cdots P\, k \cdots\ \texttt{===>}\ Q}{\cdots (\texttt{sepex}\ k \texttt{,} P\, k) \cdots\ \texttt{===>}\ Q}$$

`Ent_R_ex n c`

$$\frac{P\ \texttt{===>}\ \cdots Q\, c \cdots}{P\ \texttt{===>}\ \cdots (\texttt{sepex}\ i \texttt{,} Q\, i) \cdots}$$

6.2.3 Logical Rules with Pure Assertions

`Ent_L_dup (P1 :: P2 :: nil)`

$$\frac{\cdots \star (P_1 \star P_1) \star \cdots \star (P_2 \star P_2) \star \cdots \ \texttt{===>}\ Q}{\cdots \star P_1 \star \cdots \star P_2 \star \cdots \ \texttt{===>}\ Q}$$

where $P1$ and $P2$ are bbang or sbang assertions.
`Ent_L_contract_bbang i`

$$\frac{P_0 \star \cdots \star P_n \ \texttt{===>}\ Q}{P_0 \star \cdots \star P_i \star \cdots \star P_n \ \texttt{===>}\ Q}$$

where P_i is of the form `` `! `` b.
`Ent_L_contradict (P1 :: P2 :: nil)`

$$\frac{}{\cdots \star \texttt{`!}\ ({}^b\ e) \star \cdots \star \texttt{`!}\ {}^b\neg({}^b\ e) \star \cdots \ \texttt{===>}\ Q}$$

where `P1` is `` `! `` $({}^b\ e)$ and `P2` is `` `! `` ${}^b\neg({}^b\ e)$ (or vice-versa). `Ent_L_contradict_idx l` operates with a list of indices `l`.

`Ent_L_sbang_all`

repeated application of $\dfrac{\mathcal{P} \to R \ \texttt{===>}\ Q}{\texttt{!!}\ \mathcal{P} \star \mathcal{R} \ \texttt{===>}\ \mathcal{Q}}$

`Ent_R_sbang_all`

repeated application of $\dfrac{\mathcal{P} \qquad R \ \texttt{===>}\ Q}{R \ \texttt{===>}\ \texttt{!!}\ \mathcal{P} \star \mathcal{Q}}$

6.2.4 Rewriting and Substitution

`Ent_LR_rewrite_eq_e n` where `n` is the occurrence number of an assertion (`` `! `` str = e) in the lhs, e.g.:

$$\frac{P_1\texttt{\{}str\ \texttt{<-}\ e\texttt{\}} \star P_2\texttt{\{}str\ \texttt{<-}\ e\texttt{\}} \ \texttt{===>}\ Q\texttt{\{}str\ \texttt{<-}\ e\texttt{\}}}{P_1 \star (\texttt{`!}\ str = e) \star P_2 \ \texttt{===>}\ Q}$$

`Ent_LR_rewrite_eq_p n` is when the assertion involves a pointer equality. `Ent_R_rewrite_eq_e n` rewrite only on the right of the entailment.
`Ent_R_subst_con_distr`

$$\frac{P \ \texttt{===>}\ \bigstar_i(Q_i\texttt{\{}v\ \texttt{<-}\ str\texttt{\}})}{P \ \texttt{===>}\ (\bigstar_i Q_i)\texttt{\{}v\ \texttt{<-}\ str\texttt{\}}}$$

`Ent_LR_subst_apply` applies the first visibe occurrence of a substitution on a basic assertion, by applying, among other, the following rules:

`wp_assign_bbang`

$$\frac{P \ \texttt{===>}\ \texttt{`!}\ b\{e/str\}}{P \ \texttt{===>}\ \texttt{`!}\ b\texttt{\{}e\ \texttt{<-}\ str\texttt{\}}}$$

`wp_assign_sbang`

$$\frac{P \ \texttt{===>}\ \texttt{!!}\ \mathcal{P}}{P \ \texttt{===>}\ \texttt{!!}\ \mathcal{P}\texttt{\{}e\ \texttt{<-}\ str\texttt{\}}}$$

$\cdots$

`Ent_L_subst_apply` operates on the lhs, `Ent_R_subst_apply` on the rhs, `Ent_L_subst_apply_bbang_occ n` on the `n` occurrence of (`` `! `` b).
`Ent_LR_subst_inde` eliminates the substitution over a basic assertion when the latter does not depend on the former, e.g.:

$$\frac{v \mapsto_{\mathrm{lV}} lv \ \texttt{===>}\ Q}{(v \mapsto_{\mathrm{lV}} lv)\texttt{\{}str\ \texttt{<-}\ v_1\texttt{\}} \ \texttt{===>}\ Q} \qquad \frac{str \text{ does not appear in } P \qquad R \ \texttt{===>}\ (\texttt{sepex}\ i\texttt{,}P\ i)}{R \ \texttt{===>}\ (\texttt{sepex}\ i\texttt{,}P\ i)\texttt{\{}str\ \texttt{<-}\ e\texttt{\}}}$$

6.3 Tactics About Hoare Triples

6.3.1 Enhanced Inference Rules

`Hoare_ifte_bang H`

$$\frac{\{\, P \star \text{‘}\,\texttt{!}\; b \,\}\; c_1 \;\{\, Q \,\} \qquad \{\, P \star \text{‘}\,\texttt{!}\;{}^{b}\neg b \,\}\; c_2 \;\{\, Q \,\}}{\{\, P \,\}\; \texttt{If}\; b\, \texttt{Then}\; c_1\, \texttt{Else}\; c_2\; \{\, Q \,\}}$$

where the Coq variable `H` is set to ‘`!` b or ‘`!` ${}^{b}\neg b$ according to the branch taken.
`Hoare_seq_ext A`

$$\frac{\{\, P \,\}\; c_1 \;\{\, P \star A \,\} \qquad \{\, P \star A \,\}\; c_2 \;\{\, Q \,\}}{\{\, P \,\}\; c_1 \;\; \texttt{;} \;\; c_2 \;\{\, Q \,\}}$$

6.3.2 Structural Rules

`Hoare_L_stren_by P l`

$$\frac{l \subseteq R \qquad l \; \texttt{===>} \; P \qquad \{\, P \star R \,\}\; c \;\{\, Q \,\}}{\{\, R \,\}\; c \;\{\, Q \,\}}$$

`Hoare_seq_replace1 P1 P2`

$$\frac{\{\, \cdots \star P_1 \star \cdots \,\}\; c_1 \;\{\, \cdots \star P_2 \star \cdots \,\} \qquad \{\, \cdots \star P_2 \star \cdots \,\}\; c_2 \;\{\, Q \,\}}{\{\, \cdots \star P_1 \star \cdots \,\}\; c_1 \;\; \texttt{;} \;\; c_2 \;\{\, Q \,\}}$$

`Hoare_seq_replace l1 l2`

$$\frac{l_1 \subseteq P \qquad \{\, P \,\}\; c_1 \;\{\, (P \backslash l_1) \star l_2 \,\} \qquad \{\, (P \backslash l_1) \star l_2 \,\}\; c_2 \;\{\, Q \,\}}{\{\, P \,\}\; c_1 \;\; \texttt{;} \;\; c_2 \;\{\, Q \,\}}$$

`Hoare_frame l1 l2`

$$\frac{l_1 \subseteq P \qquad l_2 \subseteq Q \qquad c \text{ does not depend on } P \backslash l_1 = Q \backslash l_2 \qquad \{\, l_1 \,\}\; c \;\{\, l_2 \,\}}{\{\, P \,\}\; c \;\{\, Q \,\}}$$

Variant: `Hoare_frame_idx l1 l2` where `l1` and `l2` are lists of indices only keeps assertions whose index is in `l1` or `l2`.

6.3.3 Logical Rules

Here, `n`th is the occurrence number of the logical connective ($\vee$, $\exists$).
`Hoare_L_or n`

$$\frac{\{\, \cdots P_1 \cdots \,\}\; c \;\{\, Q \,\} \qquad \{\, \cdots P_2 \cdots \,\}\; c \;\{\, Q \,\}}{\{\, \cdots P_1 \vee P_2 \cdots \,\}\; c \;\{\, Q \,\}}$$

`Hoare_L_ex n i`

$$\frac{\forall i, \{\, \cdots P i \cdots \,\}\; c \;\{\, Q \,\}}{\{\, \cdots (\texttt{sepex}\; i, P i) \cdots \,\}\; c \;\{\, Q \,\}}$$

6.3.4 Logical Rules with Pure Assertions

`Hoare_L_dup (P1 :: P2 :: nil)`

$$\frac{\{\ \cdots \star (P_1 \star P_1) \star \cdots \star (P_2 \star P_2) \star \cdots\ \}\ c\ \{\ Q\ \}}{\{\ \cdots \star P_1 \star \cdots \star P_2 \star \cdots\ \}\ c\ \{\ Q\ \}}$$

where $P1$ and $P2$ are bbang or sbang assertions.

`Hoare_L_contract_bbang P2`

$$\frac{\{\ P_1 \star P_3\ \}\ c\ \{\ Q\ \}}{\{\ P_1 \star P_2 \star P_3\ \}\ c\ \{\ Q\ \}}$$

when $P_2 = \texttt{`!}\ b$ for some b.

`Hoare_L_contradict (i1 :: i2 :: nil)`

$$\frac{}{\{\ \cdots \star \texttt{`!}\ ({}^b\ e) \star \cdots \star \texttt{`!}\ {}^b\neg({}^b\ e) \star \cdots\ \}\ c\ \{\ Q\ \}}$$

where `i1` is the index of $\texttt{`!}\ ({}^b\ e)$ and `i2`, the index of $\texttt{`!}\ {}^b\neg({}^b\ e)$.

`Hoare_R_sbang n`

$$\frac{\mathcal{P} \qquad \{\ P\ \}\ c\ \{\ \cdots \star \cdots\ \}}{\{\ P\ \}\ c\ \{\ \cdots \star \texttt{!!}\ \mathcal{P} \star \cdots\ \}}$$

`Hoare_L_sbang n`

$$\frac{P \to \{\ \cdots \star \cdots\ \}\ c\ \{\ Q\ \}}{\{\ \cdots \star \texttt{!!}\ \mathcal{P} \star \cdots\ \}\ c\ \{\ Q\ \}}$$

for the `n`th occurrence of the pure assertion.

References

[1] Reynald Affeldt and Kazuhiko Sakaguchi. An Intrinsic Encoding of a Subset of C and its Application to TLS Network Packet Processing. Journal of Formalized Reasoning, 7(1):6 (2014)

[2] Reynald Affeldt and Nicolas Marti. Towards Formal Verification of TLS Network Packet Processing Written in C. In 7th ACM SIGPLAN Workshop on Programming Languages meets Program Verification (PLPV 2013), January 22, 2013, Rome, Italy, pages 35-46. ACM, January 2013.

[3] A Library for Formal Verification of Low-level Programs. `https://staff.aist.go.jp/reynald.affeldt/coqdev/`