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
   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
1 A Subset of the C Language

1.1 Values

\(\langle x \rangle_n\) is notation for a finite-size (n bits) unsigned integer with decimal value \(x\) (if it exists). \(\langle x \rangle_s\) is for signed integers. A physical value of type \(t\) is a sequence of bytes \(l\) such that \(|l| = \text{sizeof}(t)\). 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 \(pv_0\). 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 \(pv \cong lv\).

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. \(E[e]_s\) 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>
<thead>
<tr>
<th>Numerical operators</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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 \times e_2)</td>
<td>arithmetic multiplication</td>
</tr>
<tr>
<td>(e_1 &amp; e_2)</td>
<td>bitwise and</td>
</tr>
<tr>
<td>(e_1 | e_2)</td>
<td>bitwise or</td>
</tr>
<tr>
<td>(e_1 \ll e_2)</td>
<td>left-shift</td>
</tr>
<tr>
<td>(e % n)</td>
<td>modulo (2^n)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Relational operators (returns 0 or 1, as an unsigned int)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(e_1 = e_2)</td>
<td>equality between arithmetic types or pointers, depending on the type</td>
</tr>
<tr>
<td>(e_1 \neq e_2)</td>
<td>inequality test</td>
</tr>
<tr>
<td>([\langle, \langle=, \rangle&gt;, &gt;=] e_2)</td>
<td>arithmetic comparisons</td>
</tr>
<tr>
<td>(e_1 &amp;&amp; e_2)</td>
<td>logical and</td>
</tr>
<tr>
<td>(e_1 || e_2)</td>
<td>logical or</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Others</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(e \rightarrow 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 \rightarrow 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>{class} (e)</td>
<td>unsafe cast</td>
</tr>
<tr>
<td>{unsigned} (e)</td>
<td>e.g., unsafe cast to unsigned int</td>
</tr>
<tr>
<td>NULL</td>
<td>pointers of value 0</td>
</tr>
</tbody>
</table>

Table 1: C expressions

\(b\) ranges over boolean expressions. Boolean expressions are of the form \(b \) or \(\neg b\). \(B[b]_s\) is the boolean
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 $[b]_b$.

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" ;
"y" <-* %"y" ;
%"x" *<- %"y" ;
%"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 $store \rightarrow heap \rightarrow Prop$ (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

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F$</td>
<td>never holds</td>
</tr>
<tr>
<td>$T$</td>
<td>holds for any heap</td>
</tr>
<tr>
<td>$\epsilon$</td>
<td>holds for the empty heap</td>
</tr>
<tr>
<td>$*$</td>
<td>separating conjunction</td>
</tr>
<tr>
<td>$\lor, \land$</td>
<td>lifting of the Coq $\lor$ and $\land$</td>
</tr>
<tr>
<td>$(\text{sepex } i, P i)$</td>
<td>holds when there exists an $i$ such that $P i$ holds</td>
</tr>
<tr>
<td>$a \mapsto \gammaV v$</td>
<td>maps an address (physical value) $a$ to a logical value $v$</td>
</tr>
<tr>
<td>$e \mapsto \gammaL v$</td>
<td>maps an address (expression) $e$ to a logical value $v$</td>
</tr>
<tr>
<td>$e \mapsto l$</td>
<td>holds when $e$ points to the list of physical values $l$</td>
</tr>
<tr>
<td>$e \mapsto \mathbf{T}$</td>
<td>holds when $e \mapsto l$ holds and $(! (E[l]_s) + \text{sizeof}(l) \times</td>
</tr>
</tbody>
</table>

Pure Assertions

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$! P$</td>
<td>holds when $P$ holds for the store and the heap is empty ($P$ has type store $\rightarrow$ Prop)</td>
</tr>
<tr>
<td>$^i! b$</td>
<td>holds when $^i! B[b]$ holds</td>
</tr>
<tr>
<td>$!! P$</td>
<td>holds when $P$ holds and the heap is empty</td>
</tr>
</tbody>
</table>

Substitutions

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P^{str \leftarrow c}$</td>
<td>holds when the occurrences of str in $P$ are replaced with $c$</td>
</tr>
<tr>
<td>$P^{str \leftarrow *c}$</td>
<td>holds when str is replaced with $*c$ in $P$</td>
</tr>
<tr>
<td>$P^{str \leftarrow e &amp;\rightarrow f}$</td>
<td>holds when str is replaced with $e$ &amp;\rightarrow $f$ in $P$</td>
</tr>
<tr>
<td>$P^{str \leftarrow l{ei,e,i}}$</td>
<td>holds when str is replaced with the $i$th element of array $l$ in $P$</td>
</tr>
<tr>
<td>$P^{fit{}str \leftarrow l{ei,e,i}}$</td>
<td>holds when str is replaced with the $i$th element of array $l$ in $P$ (with the additional constraint that the array fits in memory)</td>
</tr>
</tbody>
</table>

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

\[
P{x \leftarrow *e} \equiv \exists pv, s, h. h(E[e]_s) = pv \land P{s{pv/str}} h
\]
\[
P{str \leftarrow e &\rightarrow f} \equiv \exists lvs, pv, lv, s, h. lvs_f = lv \land pv \lor lv \land (e \mapsto \gammaL lvs \land \mathbf{T}) s h \land (P{pv/str}) s h
\]
\[
P{str \leftarrow l\{ei,e,i\}} \equiv \exists s, h. i < |l| \land ((^i! (b\ e_i = e + i)) \star (e \mapsto l) \star \mathbf{T}) s h \land (P{str \leftarrow l_i}) s h
\]
\[
P^{fit\{}str \leftarrow l\{ei,e,i\}} \equiv \exists s, h. i < |l| \land ((^i! (b\ e_i = e + i)) \star (e \mapsto l) \star \mathbf{T}) s h \land (P{str \leftarrow l_i}) s h
\]

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 $P' \implies Q'$ (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

\[
\{ \"x\" \mapsto \text{le \ a} \} \quad \text{swap} \quad \{ \"x\" \mapsto \text{le \ b} \}
\]

\[
\{ \"x\" \mapsto \text{le \ a} \} \quad \text{z \ \leftarrow \ \star \ \text{\{\}\{\}\}} \quad \text{other subgoals}
\]

\[
\{ \"x\" \mapsto \text{le \ a} \} \quad \{ \{\! \text{z = a}\} \text{\{\\} \mapsto \text{le \ a} \}\}
\]

\[
\text{tactics Hoare_seq_ext '! \ z = a Hoare_Frame_idx (0 :: nil) (0 :: 1 :: nil)}
\]

\[
\text{lemma hoare_lookup_stren}
\]

\[
\text{\{\}'! \ z = a \mapsto \text{le \ a} \}\{\ z \ \leftarrow \ \star \ \text{\{\}\}}
\]

\[
\text{tactics Ent_R_lookup_trans}
\]

\[
\text{lemma ent_R Lookup_trans}
\]

\[
\text{first subgoal proved}
\]

4 Naming Convention for Lemmas and Tactics

<table>
<thead>
<tr>
<th>con*</th>
<th>lemma about the separating conjunction * of the form \ldots \implies \ldots</th>
</tr>
</thead>
<tbody>
<tr>
<td>ent_L_*</td>
<td>lemma with conclusion \cdot \implies \cdot acting on the left</td>
</tr>
<tr>
<td>ent_R_*</td>
<td>lemma with conclusion \cdot \implies \cdot acting on the right</td>
</tr>
<tr>
<td>hoare_L_*</td>
<td>lemma with conclusion { \cdot } \cdot { \cdot } acting on the precondition</td>
</tr>
<tr>
<td>hoare_R_*</td>
<td>lemma with conclusion { \cdot } \cdot { \cdot } acting on the postcondition</td>
</tr>
<tr>
<td>Ent_*</td>
<td>tactic for goals \cdot \implies \cdot</td>
</tr>
<tr>
<td>Hoare_*</td>
<td>tactic for goals { \cdot } \cdot { \cdot }</td>
</tr>
</tbody>
</table>

5 Lemmas

5.1 Manipulation of Assertions

\[
\text{conA}
\]

\[
P \ast (Q \ast R) \iff (P \ast Q) \ast R \quad P \ast Q \ast R \iff Q \ast P \ast R
\]

\[
\text{conP}
\]

\[
\epsilon \ast P \iff P \quad F \ast P \iff F
\]

\[
\text{conDr}
\]

\[
R \ast (P \lor Q) \iff (R \ast P) \lor (R \ast Q) \quad (P \lor Q) \ast R \iff (P \ast R) \lor (Q \ast R)
\]

\[
\text{bbang_dup}
\]

\[
\{\! b \iff \{\! b \ast \! b\}.
\]

\[
\text{wp_assign_or}
\]

\[
(P \lor Q)\{\text{str \ \leftarrow \ e}\} \iff (P\{\text{str \ \leftarrow \ e}\}) \lor (Q\{\text{str \ \leftarrow \ e}\})
\]

5
wp_assign_ex
(sepex_i,Pi){str <-> e} <=> (sepex_i,(P_i){str <-> e})

wp_lookup_fldp_ex
(sepex_x,P x){str <-> e &-> f} <=> (sepex_x,(P x){str <-> e &-> f})

5.2 Lemmas about Entailment
5.2.1 Structural Rules

\[
\begin{align*}
\text{monotony} & : P \implies Q \\ 
\text{monotony}_L & : R \implies S \\ 
\text{monotony}_R & : P \implies Q
\end{align*}
\]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(P)</td>
<td>(\implies)</td>
<td>(Q)</td>
</tr>
<tr>
<td>(R)</td>
<td>(\implies)</td>
<td>(S)</td>
</tr>
<tr>
<td>(P)</td>
<td>(\implies)</td>
<td>(Q)</td>
</tr>
</tbody>
</table>

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

5.2.2 Logical Rules

\[
\begin{align*}
\text{ent}_L_F & : F \implies Q \\ 
\text{ent}_R_T & : P \implies T \\ 
\text{ent}_id & : P \implies P \\ 
\text{ent_trans} & : P \implies Q \quad Q \implies R
\end{align*}
\]

\[
\begin{align*}
\text{ent}_R\_or\_1 & : P \implies R \quad Q \implies R \\ 
\text{ent}_R\_or\_2 & : P \implies Q \quad R \implies Q
\end{align*}
\]

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

5.2.3 Logical Rules with Pure Assertions

\[
\begin{align*}
\text{ent}_bang\_contract & \quad \text{Variants: ent_bbang}\_contract \quad \text{ent}_L\_bbang \quad \text{ent}_L\_con\_bbang \quad \text{ent}_L\_bbang\_con \\
! P & \implies \epsilon \\
\epsilon & \implies ! b \\
\epsilon & \implies R \\
P \implies ! b & \implies R \\
P \implies R & \\
\epsilon & \implies ! \epsilon \\
\epsilon & \implies ! ! P \\
\epsilon & \implies ! ! Q
\end{align*}
\]

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

\[
\begin{align*}
\text{ent}_R\_sbang & \\
\epsilon & \implies ! ! P \\

\text{ent}_L\_sbang\_con & \\
P \implies Q & \implies R \\
!! P \implies Q & \implies R
\end{align*}
\]

\[
\begin{align*}
\text{ent}_sbang\_sbang & \\
\epsilon & \implies ! ! P \\

\text{ent}_R\_sbang & \quad \text{Variant: ent}_R\_sbang\_con \\
\epsilon & \implies ! ! P \\

\text{ent}_sbang\_sbang & \quad \text{Variant: ent}_R\_sbang\_con \\
P \implies Q & \implies R \\
!! P \implies Q & \implies R
\end{align*}
\]
5.2.4 Turn Substitutions into Substitutions in the Store

\[
\begin{align*}
\text{ent}_R\_lookup\_trans & : \begin{array}{l}
      pv \odot lv \quad R \Longrightarrow e \mapsto_{\gamma} lv \ast T \\
      \end{array} \\
& \frac{}{R \Longrightarrow P\{str <- \ [pv]_e\}} \\
\text{ent}_R\_lookup\_fldp & : \begin{array}{l}
      pv \odot lvs_f \quad e \mapsto_{\gamma} lvs \Longrightarrow P\{str <- \ [pv]_e\} \\
      \end{array} \\
& \frac{}{e \mapsto_{\gamma} lvs \Longrightarrow P\{str <- e \mapsto f\}}
\end{align*}
\]

\[
\begin{align*}
\text{ent}_R\_lookup\_fldp\_trans & : \begin{array}{l}
      pv \odot lvs_f \quad R \Longrightarrow e \mapsto_{\gamma} lvs \ast T \\
      \end{array} \\
& \frac{}{R \Longrightarrow P\{str <- [pv]_e\}}
\end{align*}
\]

\[
\begin{align*}
\text{ent}_R\_lookup\_mapstos\_trans & : \begin{array}{l}
      i < |l| \quad R \Longrightarrow '!' \{b \quad e_i = e + i\} \ast (e \mapsto l) \ast T \\
      \end{array} \\
& \frac{}{R \Longrightarrow P\{str <- l\{e_i, e, i\}\}}
\end{align*}
\]

\[
\begin{align*}
\text{ent}_R\_lookup\_mapstos\_fit\_trans & : \begin{array}{l}
      i < |l| \quad R \Longrightarrow '!' \{b \quad e_i = e + i\} \ast (e \mapsto l) \ast T \\
      \end{array} \\
& \frac{}{R \Longrightarrow P^{\mathit{fit}}\{str <- l\{e_i, e, i\}\}}
\end{align*}
\]

5.3 Lemmas about Hoare Triples

5.3.1 Structural Rules

\[
\text{hoare\_stren} : \begin{array}{l}
      P \Longrightarrow P' \quad \{P\} c \{Q\} \\
      \end{array} \\
& \frac{}{\{P\} c \{Q\}}
\]

\[
\text{hoare\_weak} : \begin{array}{l}
      P \Longrightarrow Q' \quad Q \Longrightarrow Q' \\
      \end{array} \\
& \frac{}{\{P\} c \{Q\}}
\]

5.3.2 Logical Rules

\[
\text{hoare\_L\_F} : \begin{array}{l}
      \{F\} c \{Q\} \\
      \end{array} \\
& \frac{}{\{P \land P'\} c \{Q \land Q'\}}
\]

\[
\text{hoare\_LR\_and} : \begin{array}{l}
      \{P\} c \{Q\} \quad \{P'\} c \{Q'\} \\
      \end{array} \\
& \frac{}{\{P \land P'\} c \{Q \land Q'\}}
\]

\[
\text{hoare\_L\_or} : \begin{array}{l}
      \{P_1\} c \{Q\} \quad \{P_2\} c \{Q\} \\
      \end{array} \\
& \frac{}{\{P_1 \lor P_2\} c \{Q\}}
\]

\[
\text{hoare\_R\_or\_1} : \begin{array}{l}
      \{P\} c \{Q_1\} \\
      \end{array} \\
& \frac{}{\{P\} c \{Q_1 \lor Q_2\}}
\]

\[
\text{hoare\_R\_or\_2} : \begin{array}{l}
      \{P\} c \{Q_2\} \\
      \end{array} \\
& \frac{}{\{P\} c \{Q_1 \lor Q_2\}}
\]

NB: The tactic \text{Hoare\_L\_or} (see Sect. 6.3) generalizes \text{hoare\_L\_or}.

\[
\text{hoare\_R\_ex} : \begin{array}{l}
      \exists x, \{P\} c \{Q\} \\
      \end{array} \\
& \frac{}{\{P\} c \{\exists x, Q\ \}}
\]

\[
\text{hoare\_L\_ex} : \begin{array}{l}
      \forall x, \{P\} c \{Q\} \\
      \end{array} \\
& \frac{}{\{P\} c \{\exists x, Q\ \}}
\]

NB: The tactic \text{Hoare\_L\_ex} (see Sect. 6.3) generalizes \text{hoare\_L\_ex}.
5.3.3 Inference Rules

Basic Commands

\[
\text{hoare0 Assign} \\
\{ P\{\text{str} \leftarrow e \} \} \text{str} \leftarrow e \{ P \}
\]

\[
\text{hoare0 Lookup} \\
\{ P\{x \leftarrow \ast e \} \} x \leftarrow \ast e \{ P \}
\]

Composition of Commands

\[
\text{hoare Seq} \\
\{ P \} c_1 \{ R \} \{ R \} c_2 \{ Q \} \\
\{ P \} c_1 ; c_2 \{ Q \}
\]

\[
\text{hoare Ifte Bang} \\
\{ P \ast \ast b \} c_1 \{ Q \} \{ P \ast \ast b \times b \} c_2 \{ Q \} \\
\{ P \} \text{If } b \text{Then } c_1 \text{Else } c_2 \{ Q \}
\]

\[
\text{hoare Forloop2} \\
\{ P \} c_1 ; c_2 \{ \text{Inv} \} \{ \text{Inv} \ast \ast \} \{ P \} \text{For}(c_1;c_2;e;c)\{\{e\}\} \{ Q \}
\]

Variants of Basic Commands

\[
\text{hoare Mutation Local Forward Ground le} \\
e_2 \text{ has no free variable} \\
\{ e_1 \rightarrow le \} \text{lv} \times e_2 \{ e_1 \rightarrow le \text{lv}' \}
\]

Array Access

\[
\text{hoare Lookup Mapstos} \\
t \text{is the type of str} \\
l \text{is a list of physical values of type t} \\
e_i, e \text{ are expressions of type } \ast t \\
\|l\| \times \text{sizeof}(t) < 2^{31} \\
\{ P\{\text{str} \leftarrow l(e_i,e_i)\} \} \text{str} \leftarrow \ast e_i \{ P \}
\]

\[
\text{hoare Lookup Mapstos Fit} \\
t \text{is the type of str} \\
l \text{is a list of physical values of type t} \\
e_i, e \text{ are expressions of type } \ast t \\
\|l\| \times \text{sizeof}(t) < 2^{31} \\
\{ P\{\text{str} \leftarrow l(e_i,e_i)\} \} \text{str} \leftarrow \ast e_i \{ P \}
\]

Structure Access
hoare_lookup_fldp

\{ P \{ str \leftarrow e \land \rightarrow f \} \} \ str \leftarrow* e \land \rightarrow f \ \{ P \} \\

variant: hoare_lookup_fldp_stren

\{ P \ \{ str \leftarrow e \land \rightarrow f \} \} \ P \implies Q \{ str \leftarrow e \land \rightarrow f \} \ \{ Q \}

hoare_mutation_fldp_local_forward_ground_1v

\{ pe \implies \nu e lvs \} \ pe \land \rightarrow f \ \iff e \ \{ pe \implies \nu e lvs(lv/f) \}

hoare_mutation_fldp_local_forward_ground_le

\{ e_1 \implies \nu e lvs \} \ e_1 \land \rightarrow f \ \iff e_2 \ \{ e_1 \implies \nu e lvs(lv/f) \}

hoare_mutation_fldp_subst_ityp/ptr

\{ P \implies str = e \} \ e_1 \{ e/str \} \land \rightarrow f \ \iff e_2 \{ e/str \} \ \{ Q \}

\{ e_1 \{ e/str \} \land \rightarrow f \ \iff e_2 \{ e/str \} \ \{ Q \}

6 Tactics

6.1 Manipulation of Assertions

Bbang2sbang turns all the occurrences of '!' b into '!! [b]gb when there is no free variable in b.

6.2 Tactics about Entailment

6.2.1 Structural Rules

\begin{align*}
\text{Ent}_L\text{.stren by P 1} & \\
\frac{l \subseteq R \quad l \implies P \quad P \land \land R \implies Q}{R \implies Q}
\end{align*}

\text{Ent.permut}:

\begin{align*}
\frac{\sigma \text{ permutation}}{\star_i P_i \implies \star_{\sigma(i)} P_i}
\end{align*}

\text{Ent.decompose 11 12}:

\begin{align*}
\star_{i \in l_1} P_i \implies \star_{j \in l_2} Q_j \\
\star_{i \in l_1} P_i \implies \star_{j \in l_2} Q_j
\end{align*}

\text{Ent.monotony} simplifies \( P \implies Q \) by eliminating assertions that appear in both \( P \) and \( Q \). \text{Ent.monotony0} is a faster but simpler version of \text{Ent.monotony}, e.g., it solves \( P \implies P \land (\lor b = b) \).

6.2.2 Logical Rules

Here, \( n \) is the occurrence number of the logical connective (\( \lor, \exists \)).

\begin{align*}
\text{Ent_L.or n} & \\
\frac{\ldots P \ldots \implies Q \quad \ldots R \ldots \implies Q}{\ldots P \lor R \ldots \implies Q}
\end{align*}

\begin{align*}
\text{Ent_R.or.1 n} & \\
\frac{P \implies \ldots Q \ldots}{P \implies \ldots Q \lor R \ldots}
\end{align*}

\begin{align*}
\text{Ent_R.or.2 n} & \\
\frac{P \implies \ldots R \ldots}{P \implies \ldots Q \lor R \ldots}
\end{align*}

\begin{align*}
\text{Ent_L.ex n k} & \\
\frac{\forall k, \ldots P k \ldots \implies Q}{\ldots (\text{sepex } k, P k) \ldots \implies Q}
\end{align*}

\begin{align*}
\text{Ent_R.ex n c} & \\
\frac{P \implies \ldots Q c \ldots}{P \implies \ldots (\text{sepex } i, Q i) \ldots}
\end{align*}
6.2.3 Logical Rules with Pure Assertions

Ent_L_dup (P1 :: P2 :: nil)

\[
\cdots \star (P_1 \star P_1) \star \cdots \star (P_2 \star P_2) \cdots \implies Q
\]

\[
\cdots \star P_1 \star \cdots \star P_2 \star \cdots \implies Q
\]

where \( P_1 \) and \( P_2 \) are bbang or sbang assertions.

Ent_L_contract_bbang i

\[
P_0 \star \cdots \star P_i \star \cdots \star P_n \implies Q
\]

where \( P_i \) is of the form \( '! b \).

Ent_L_contradict (P1 :: P2 :: nil)

\[
\cdots \star (! (b e) \star \cdots \star ! b \neg (b e) \star \cdots \implies Q
\]

where \( P_1 \) is \( '! (b e) \) and \( P_2 \) is \( '! b \neg (b e) \) (or vice-versa). Ent_L_contrdict_idx l operates with a list of indices l.

Ent_L_sbang_all

repeated application of \( \mathcal{P} \rightarrow \mathcal{R} \implies Q \)

Ent_R_sbang_all

repeated application of \( \mathcal{P} \star \mathcal{R} \implies Q \)

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.:

\[
P_1 \{ \text{str} \leftarrow e \} \star P_2 \{ \text{str} \leftarrow e \} \implies Q \{ \text{str} \leftarrow e \} \]

\[
P_1 \{ '! \text{str} = e \} \star P_2 \implies Q
\]

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

\[
P \implies \star_i (Q_i \{ v \leftarrow \text{str} \})
\]

\[
P \implies (\star_i Q_i) \{ v \leftarrow \text{str} \}
\]

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

wp_assign_bbang

\[
P \implies '! b \{ e/\text{str} \}
\]

wp_assign_sbang

\[
P \implies !! P \implies !! P \{ e \leftarrow \text{str} \}
\]

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.:

\[
v \mapsto v \mapsto Q
\]

\[
(v \mapsto v) \{ \text{str} \leftarrow v_1 \} \implies Q
\]

\[
\text{str does not appear in } P
\]

\[
R \implies (\text{sepex } i, P i)
\]

\[
R \implies (\text{sepex } i, P i) \{ \text{str} \leftarrow e \}
\]
6.3 Tactics About Hoare Triples

6.3.1 Enhanced Inference Rules

Hoare_ifte_bang H

\[
\frac{\{ P \mathrel{\star} ! b \} \ c_1 \ \{ Q \} \quad \{ P \mathrel{\star} \neg b \} \ c_2 \ \{ Q \}}{\{ P \} \text{If } b \text{Then } c_1 \text{Else } c_2 \ \{ Q \}}
\]

where the Coq variable H is set to ‘! b or ‘! b according to the branch taken.

Hoare_seq_ext A

\[
\frac{\{ P \} \ c_1 \ \{ P \mathrel{\star} A \} \quad \{ P \mathrel{\star} A \} \ c_2 \ \{ Q \}}{\{ P \} \ c_1 \ ; \ c_2 \ \{ Q \}}
\]

6.3.2 Structural Rules

Hoare_L_stren_by P l

\[
\frac{l \subseteq R \quad l \Rightarrow P \quad \{ P \mathrel{\star} R \} \ c_1 \ \{ Q \}}{\{ R \} \ c_1 \ ; \ c_2 \ \{ Q \}}
\]

Hoare_seq_replace1 P1 P2

\[
\frac{\{ \cdots \mathrel{\star} P_1 \cdots \} \ c_1 \ \{ \cdots \mathrel{\star} P_2 \cdots \} \ c_2 \ \{ Q \}}{\{ \cdots \mathrel{\star} P_1 \cdots \} \ c_1 \ ; \ c_2 \ \{ Q \}}
\]

Hoare_seq_replace l1 l2

\[
\frac{l_1 \subseteq P \quad \{ P \} \ c_1 \ \{ (P \setminus l_1) \mathrel{\star} l_2 \} \ c_2 \ \{ Q \}}{\{ P \} \ c_1 \ ; \ c_2 \ \{ Q \}}
\]

Hoare_frame l1 l2

\[
\frac{l_1 \subseteq P \quad l_2 \subseteq Q \quad \text{c does not depend on } P \setminus l_1 = Q \setminus l_2 \quad \{ l_1 \} \ c_1 \ \{ l_2 \}}{\{ P \} \ c \ \{ Q \}}
\]

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, nth is the occurrence number of the logical connective (\lor, \exists).

Hoare_L_or n

\[
\frac{\{ \cdots P_1 \cdots \} \ c \ \{ Q \} \quad \{ \cdots P_2 \cdots \} \ c \ \{ Q \}}{\{ \cdots P_1 \lor P_2 \cdots \} \ c \ \{ Q \}}
\]

Hoare_L_ex n i

\[
\frac{\forall i, \{ \cdots P_i \cdots \} \ c \ \{ Q \}}{\{ \cdots (\text{sepex } i, P_i) \cdots \} \ c \ \{ Q \}}
\]
6.3.4 Logical Rules with Pure Assertions

\text{Hoare\_L\_dup} (P_1 :: P_2 :: \text{nil})
\[
\frac{\{ \cdots \star (P_1 \star P_1) \cdots \star (P_2 \star P_2) \cdots \} \ c \ \{ \ Q \ \}}{\{ \cdots \star P_1 \cdots \star P_2 \cdots \} \ c \ \{ \ Q \ \}}
\]

where \(P_1\) and \(P_2\) are \text{bbang} or \text{sbang} assertions.

\text{Hoare\_L\_contract\_bbang} P_2
\[
\frac{\{ P_1 \star P_3 \} \ c \ \{ \ Q \ \}}{\{ P_1 \star P_2 \star P_3 \} \ c \ \{ \ Q \ \}}
\]

when \(P_2 = \star \ b\) for some \(b\).

\text{Hoare\_L\_contradict} (i_1 :: i_2 :: \text{nil})
\[
\frac{\{ \cdots \star \ (b \ e) \cdots \star \ (\ b \ ^{-} (b \ e)) \cdots \} \ c \ \{ \ Q \ \}}{}
\]

where \(i_1\) is the index of \(\star \ (b \ e)\) and \(i_2\), the index of \(\star \ (\ b \ ^{-} (b \ e))\).

\text{Hoare\_R\_sbang} n \text{ Hoare\_L\_sbang} n
\[
\frac{\{ \ P \ \} \ c \ \{ \ \cdots \ \star \cdots \ \} \ }{\{ \ P \ \} \ c \ \{ \ \cdots \star \ !! \ P \cdots \ \} \}
\]

\text{P} \rightarrow \{ \ \cdots \ \star \cdots \ \} \ c \ \{ \ Q \ \}
\]

for the \(n\)th occurrence of the pure assertion.

References

