Gradually Verified Language with Recursive Predicates

Henry Blanchette

Contents

1 Weakest Predonditions

1.1 Concrete Weakest Liberal Precondition (WLP) Rules

$\mathsf{WLP} : \textsc{Statement} \times \textsc{SatFormula} \to \textsc{SatFormula}$

$\mathsf{WLP}(s, \phi) :=$ match s with

`skip`	$\mapsto$	ϕ
s_1 `;` s_2	$\mapsto$	$\mathsf{WLP}(s_1,\ \mathsf{WLP}(s_2, \phi))$
$T\ x$	$\mapsto$	ϕ
x `:=` e	$\mapsto$	$\mathsf{footprint}(e)\ \wedge\ [e/x]\phi$
x `:=` `new` C	$\mapsto$	$[\mathsf{new}(C)/x]\phi$
$x.f$ `:=` y	$\mapsto$	$\mathsf{footprint}(x.f)\ \wedge\ [y/x.f]\phi$
y `:=` $z.m_C(\overline{e})$	$\mapsto$	$\mathsf{footprint}(\overline{e})\ \wedge\ z$ `!=` `null` $\wedge$
		$[z/\texttt{this},\ \overline{e/x}]\mathsf{pre}(m_C)\ *$
		$\mathsf{handleMethod}(m_C, \phi)$
`if` (e) $\{s_{\text{the}}\}$ `else` $\{s_{\text{els}}\}$	$\mapsto$	`if` (e) `then` $\mathsf{WLP}(s_{\text{the}}, \phi)$
		`else` $\mathsf{WLP}(s_{\text{els}}, \phi)$
`while` (e) `invariant` ϕ_{inv} $\{s_{\text{bod}}\}$	$\mapsto$	$\mathsf{footprint}(e)\ \wedge\ \phi_{\text{inv}}\ \wedge$
		`if` e `then` $\mathsf{WLP}(s_{\text{bod}}, \phi_{\text{inv}})$ `else` $\mathsf{WLP}(\texttt{true}, \phi)$
`assert` ϕ_{ass}	$\mapsto$	$\mathsf{footprint}(\phi_{\text{ass}})\ \wedge\ \phi_{\text{ass}}\ \wedge\ \phi$
`hold` ϕ_{hol} $\{s_{\text{bod}}\}$	$\mapsto$	(unimplemented)
`release` ϕ_{rel}	$\mapsto$	(unimplemented)
`unfold` $\alpha_C(\overline{e})$	$\mapsto$	$\mathsf{footprint}(\overline{e})\ \wedge$
		$[\mathsf{unfolded}(\alpha_C(\overline{e}))/\alpha_C(\overline{e}),$
		$\phi'/\texttt{unfolding}\ \alpha_C(\overline{e})\ \texttt{in}\ \phi']\phi$
`fold` $\alpha_C(\overline{e})$	$\mapsto$	$\mathsf{footprint}(\overline{e})\ \wedge\ [\alpha_C(\overline{e})/\mathsf{unfolded}(\alpha_C(\overline{e}))]\phi$

1.1.1 Utility Functions

The implementations of the functions in this section can be made much more efficient than the naive definition here in mathematical notation. For example, calculating the footprint of expressions and formulas can avoid redundancy by not generating permission-subformulas that are already satisfied. This can be implemented as implicit in $\wedge$ by a wrapper $\wedge_{\text{wrap}}$ operation in some way similar to this:

$$\phi \ \wedge_{\text{wrap}} \ \phi' := \begin{cases} \phi & \text{if } \phi \implies \phi' \\ \phi \wedge \phi' & \text{otherwise} \end{cases}$$

Some utility functions are defined as follows:

$$\begin{array}{lcl}
\mathsf{new}(C) & := & \text{an object that is a new instance of class } C \\
 & & \text{where all fields are assigned to their default values} \\
\mathsf{unfolded}(\alpha_C(\overline{e})) & := & [\overline{e/x}]\mathsf{body}(\alpha_C) \\
\mathsf{pre}(z.m_C(\overline{e})) & := & [z/\texttt{this},\ \overline{e/x}]\mathsf{pre}(m_C) \\
\mathsf{pre}(m_C) & := & \text{the static pre-condition of } m_C \\
\mathsf{post}(z.m_C(\overline{e})) & := & [z/\texttt{this},\ \overline{e/\texttt{old}(x)}]\mathsf{post}(m_C) \\
\mathsf{post}(m_C) & := & \text{the static post-condition of } m_C \\
\mathsf{body}(\alpha_C) & := & \text{the body of } \alpha_C
\end{array}$$

$$\begin{array}{lcll}
\mathsf{footprint}(e) & := & \text{match } e \text{ with} & \\
 & & \mid\ e.f & \mapsto\ \mathsf{footprint}(e') \ \wedge\ e' \ \texttt{!= null} \ \wedge\ \texttt{acc}(e'.f) \\
 & & \mid\ e_1 \oplus e_2 & \mapsto\ \mathsf{footprint}(e_1) \ \wedge\ \mathsf{footprint}(e_2) \\
 & & \mid\ e_1 \odot e_2 & \mapsto\ \mathsf{footprint}(e_1) \ \wedge\ \mathsf{footprint}(e_2) \\
 & & \mid\ e & \mapsto\ \texttt{true} \\
\mathsf{footprint}(\overline{e}) & := & \bigwedge \mathsf{footprint}(e) & \\
\mathsf{footprint}(\phi) & := & \bigwedge \{\mathsf{footprint}(e) : e \text{ appears in } \phi\} \ \wedge & \\
 & & \bigwedge \{\alpha_C(\overline{e}) : \texttt{unfolding}\ \alpha_C(\overline{e})\ \texttt{in}\ \phi' \text{ appears in } \phi\} &
\end{array}$$

$$\mathsf{handleMethod}(z.m_C(\overline{e}), \phi) \ := \ (\mathsf{footprint}(\mathsf{pre}(z.m_C(\overline{e}))) \ \wedge\ \mathsf{footprint}(\mathsf{post}(z.m_C(\overline{e})))) \ *\ \phi$$