Mechanised Hypersafety Proofs about Structured Data

Appendix

ANONYMOUS AUTHOR(S)

A LGTM RULES

$$\frac{S_1 \cap S_2 = \varnothing}{\mathsf{wp}\ [S_1 : \mathcal{P}_1,\ S_2 : \mathcal{P}_2]\ \{Q\} \dashv\vdash \mathsf{wp}\ [S_1 :\ \mathcal{P}_1]\ \{\overline{v} \mid \mathsf{wp}\ [S_2 :\ \mathcal{P}_2]\ \{\overline{u} \mid Q(\overline{vu})\}\}}\ \textsc{WpNest}$$

$$\frac{\mathtt{local}(\{P_i, Q_i\}, i) \qquad \forall i, \{\ \ulcorner i \in S \urcorner * P_i\ \}\ [i :\ \mathcal{P}(i)]\ \{\ x \mid Q_i(x)\ \}}{\{\textstyle\ast_{i\in S} P_i\}\ [S :\ \mathcal{P}]\ \{\overline{x} \mid \textstyle\ast_{i\in S} Q_i(\overline{x}(i))\}}\ \textsc{Product}$$

$$\frac{H \vdash H' \quad Q' \vdash Q \quad \{H'\}\ [S : \mathcal{P}]\ \{Q'\}}{\{H\}\ [S : \mathcal{P}]\ \{Q\}}\ \textsc{Conseq} \qquad \frac{\{H\}\ [S : \mathcal{P}]\ \{Q\}}{\{H' * H\}\ [S : \mathcal{P}]\ \{Q * H'\}}\ \textsc{Frame}$$

$$\frac{\{\ P\ \}\ [S_1 :\ \mathcal{P}_1]\ \{\ \overline{x} \mid H(\overline{x})\ \} \qquad \forall \overline{x},\ \{\ H(\overline{x})\ \}\ [S_2 :\ \mathcal{P}_2]\ \{\ \overline{y} \mid Q(\overline{xy})\ \}}{\{\ P\ \}\ [S_1 :\ \mathcal{P}_1,\ S_2 :\ \mathcal{P}_2]\ \{\ \overline{z} \mid Q(\overline{z})\ \}}\ \textsc{SeqU}$$

$$\textsc{SeqU1}\quad \frac{\{\ P\ \}\ [\iota :\ p_1]\ \{\ x \mid H\ \} \quad \{\ H\ \}\ [\iota :\ p_1',\ S :\ \mathcal{P}_2]\ \{\ x, \overline{z} \mid Q(\overline{z})\ \}}{\{\ P\ \}\ [\iota : p_1; p_1',\ S : \mathcal{P}_2]\ \{\ x, \overline{y} \mid Q(\overline{xy})\ \}} \qquad \textsc{SeqU2}\quad \frac{\{P\}\ [\iota :\ p_1,\ S :\ \mathcal{P}_2]\ \{\ x, \overline{z} \mid H(\overline{z})\ \} \quad \forall \overline{z},\ \{\ H(\overline{z})\ \}\ [\iota :\ p_1']\ \{\ x \mid Q(x\overline{z})\ \}}{\{\ P\ \}\ [\iota : p_1; p_1',\ S :\ \mathcal{P}_2]\ \{\ x_1, \overline{x_2} \mid Q(x_1\overline{x_2})\ \}}$$

Fig. 1. Selected structural LGTM rules

Structural Rules. We start our overview of LGTM reasoning principles by looking at its structural rules (Fig. 1) that facilitate transformations of the triples following the structure of the *index set*, thus enabling applications of the *program-driven* rules (*cf.* Sec. A).

$$\cfrac{\cfrac{\cfrac{\{P\}\ [S_1 : \mathcal{P}_1]\ \{H\} \quad H \vdash \mathsf{wp}\ [S_2 : \mathcal{P}_2]\ \{Q\}}{\{P\}\ [S_1 : \mathcal{P}_1]\ \{\mathsf{wp}\ [S_2 : \mathcal{P}_1]\ \{Q\}\}}\ \textsc{Cons}}{P \vdash \mathsf{wp}\ [S_1 : \mathcal{P}_1]\ \{\mathsf{wp}\ [S_2 : \mathcal{P}_1]\ \{Q\}\}}\ \textsc{WpNest}}{\{P\}\ [S_1 : \mathcal{P}_1,\ S_2 : \mathcal{P}_2]\ \{Q\}}$$

Fig. 2. SeqU derivation using WpNest

For example, the rule WpNest exploits the nature of the definition of LGTM triples via the wp predicate to "bring forward" a subset of the programs $\mathcal{P}_1$ in the product (for a particular subset S_1 of the index set), enabling more specialised reasoning about this set in particular. The utility of WpNest comes from its ability to derive rules such as SeqU (as well as other structural rules from the overview). Fig. 2 shows this derivation by first replacing the triple in the conclusion by its definition in terms of the weakest preconditions, then applying WpNest, and then rewriting the judgements in the wp-form back to the corresponding triples. Derivations of other rules from the overview section in the paper are similar.

We have already seen an application of the rule Product in the Section 2 of the paper. The remaining structural rules from Fig. 1 are standard for separation logic-style proof systems. For example, LGTM enjoys a familiar Frame rule that naturally extends to hyper-heaps.

2022. ACM 2475-1421/2022/1-ART1
https://doi.org/

$$\frac{}{\{\mathsf{emp}\}\ [\mathtt{return}\ \overline{v}]\ \{\overline{u} \mid \ulcorner \overline{v} = \overline{u} \urcorner\}}\ \textsc{Ret} \qquad \frac{}{\{\overline{x} \mapsto \overline{v}\}\ [!\overline{x}]\ \{\overline{u} \mid \overline{x} \mapsto \overline{v} * \ulcorner \overline{u} = \overline{v} \urcorner\}}\ \textsc{Read}$$

$$\frac{}{\{\overline{x} \mapsto \overline{v}\}\ [\overline{x} := \overline{u}]\ \{\overline{x} \mapsto \overline{u}\}}\ \textsc{Asn} \qquad \frac{}{\{\overline{x} \mapsto \overline{v}\}\ [\mathtt{free}(\overline{x})]\ \{\mathsf{emp}\}}\ \textsc{Fr} \qquad \frac{}{\{\mathsf{emp}\}\ [\mathtt{alloc}(\overline{v})]\ \{\overline{x} \mid \overline{x} \mapsto \overline{v}\}}\ \textsc{Alc}$$

$$\frac{}{\{\mathsf{emp}\}\ [\mathtt{malloc}(\overline{n})]\ \{\overline{x} \mid \mathtt{arr}(\overline{x},\ 0_{\overline{n}})\}}\ \textsc{Malloc} \qquad \frac{}{\{\mathtt{arr}(\overline{x},\ \overline{s})\}\ [\mathtt{mfree}(\overline{x})]\ \{\mathsf{emp}\}}\ \textsc{MFree}$$

$$\frac{\{H\}\ [\mathcal{P}_1]\ \{P\} \quad \forall \overline{v},\ \{P(\overline{v})\}\ [\mathcal{P}_2[\overline{v}/\overline{x}]]\ \{Q\}}{\{H\}\ [\mathtt{let}\ \overline{x} := \mathcal{P}_1\ \mathtt{in}\ \mathcal{P}_2]\ \{Q\}}\ \textsc{Let} \qquad \frac{\{H\}\ [\mathcal{P}]\ \{Q\} \quad \forall i,\ \begin{array}{l}\overline{v}(i) \Rightarrow \mathcal{P}(i) = \mathcal{P}_1(i)\\ \neg\overline{v}(i) \Rightarrow \mathcal{P}(i) = \mathcal{P}_2(i)\end{array}}{\{H\}\ [\mathtt{if}\ (\overline{v})\ \{\mathcal{P}_1\}\ \{\mathcal{P}_2\}]\ \{Q\}}\ \textsc{If}$$

$$\frac{}{\{\mathtt{arr}(\overline{x},\ \overline{s})\}\ [\mathtt{length}(\overline{x})]\ \{\overline{u} \mid \ulcorner \forall i, \overline{u}(i) = |\overline{s}(i)| \urcorner\}}\ \textsc{Len} \qquad \frac{\forall j \in S, \overline{r}(j) = \overline{s}(j)[\overline{i}(j) \mapsto \overline{v}(j)]}{\{\mathtt{arr}(\overline{x},\ \overline{s})\}\ \left[\overline{x}[\overline{i}] := \overline{v}\right]\ \{\mathtt{arr}(\overline{x},\ \overline{r})\}}\ \textsc{AsnArr}$$

$$\frac{\forall j \in S,\ 0 \le \overline{i}(j) < |\overline{s}(j)|}{\{\mathtt{arr}(\overline{x},\ \overline{s})\}\ \left[\overline{x}[\overline{i}]\right]\ \left\{\overline{u} \;\middle|\; \mathtt{arr}(\overline{x},\ \overline{s}) * \ulcorner \forall j \in S, \overline{u}(j) = \overline{s}(j)(\overline{i}(j)) \urcorner\right\}}\ \textsc{ReadArr}$$

Fig. 3. Lockstep rules

Lockstep rules. The core of any relational program logic is formed by so-called *lockstep* rules that enable reasoning about individual programs in the product or about batches of programs that share the same syntactic structure. The lockstep rules of LGTM are shown in Fig. 3. In their presentation, we follow a few syntactic conventions. First, by applying a program construction to a vector, we mean a pointwise application: for example, for a hyper-value $\overline{v}$ and two product-programs, $\mathcal{P}_1$ and $\mathcal{P}_2$, indexed by a set S, for any $i \in S$ the effect of programs is defined as follows:

$$(\mathtt{return}\ \overline{v})(i) \triangleq \mathtt{return}\ \overline{v}(i) \qquad (\mathtt{if}(\overline{v})\{\mathcal{P}_1\}\{\mathcal{P}_2\})(i) \triangleq \mathtt{if}(\overline{v}(i))\{\mathcal{P}_1(i)\}\{\mathcal{P}_2(i)\}$$

The same convention applies to hyper-heap assertions, *e.g.*, points-to assertions distribute similarly:

$$\overline{x} \mapsto \overline{v} \triangleq \mathop{\Large *}_{i \in S} \overline{x}(i) \mapsto \overline{v}(i)$$

Since lockstep rules work on syntactically same programs, we omit the index set in the rules.

Most of the lockstep rules are just trivial extensions of correspondent rules in a regular separation logic. For example, Ret rule states that result of a program product, each component of which is a `return`-statement `return` $\overline{v}$, is a hyper-value $\overline{u}$ is equal to the one we return $\overline{v}$. The Read rule states that, if we read from a family of pointers $\overline{x}$, the result $\overline{u}$ would be equal to a vector constructed out of the contents of the pointers being read (*i.e.*, $\overline{v}$), and that the hyper-heap will remain unchanged. The Asn rule says that, in order to write a value $\overline{u}$ to a vector of pointers $\overline{x}$, one has to replace their contents $\overline{v}$ with $\overline{u}$. Fr states that after deallocation of a vector of pointes, the resulting heap is empty. Alc states that the results of a family of pointer allocations are these pointers $\overline{x}$, pointing to the assigned values $\overline{v}$. Malloc, Mfree, Len, AsnArr and ReadArr, are rules for arrays extended to the hyper-case; we elaborate on the two of them. Malloc, `malloc` takes a length of an array to be allocated. The result of the allocation is filled with zeros: $0_{\overline{n}}$ denotes a vector of sequences, each of which consists of $\overline{n}(i)$ zeros. In the AsnArr rule, substitution $\overline{s}(j)[\overline{i}(j) \mapsto \overline{v}(j)]$ is defined only if $0 \le \overline{i}(j) < |\overline{s}(j)|$. The rules Let and If are straightforward.