Foundationally Sound Annotation Verifier via Control Flow Splitting

Shanghai Jiao Tong University

Litao Zhou @ SPLASH-SRC 2022
ltzhou@sjtu.edu.cn

Supervised by Qinxiang Cao
caoqinxiang@sjtu.edu.cn

Background and Motivation

Programs can be verified …

- by writing formal specifications and proofs in a theorem prover

Iris, Verified Software Toolchain, CAKEML A Verified Implementation of ML

✓ Foundationally sound
✓ Rich assertion language
✓ Flexible proof strategies
✗ Correctness properties are not clear from proof script

Interactive Program Verifiers

- by writing annotations in the source code

Dafny, VeriFast, frama-c Software Analyzers, viper

✓ More proof automation
✓ Readable proofs
✓ Straightforward to programmers
✗ Foundational soundness proof is often lacked

Annotation Verifiers

Contributions

- **A novel framework for program verification**, based on the idea of reducing large program proofs to simpler verification goals
- **A formal language for annotated programs, ClightA**, that not only introduces assertions but also addresses logical variables in the verification context
- **A control-flow-based verification splitting algorithm**, implemented in Coq and proved sound w.r.t. the VST program logic

Features of VST-A

✓ Correctness proofs are described intuitively by inserting assertions
✓ Rich assertion languages and foundational soundness of VST
✓ Assertions can be inserted in a flexible way
e.g. annotating loop structures with invariants is not compulsory
✓ Incremental verification for incremental program development
✗ Currently only supports sequential programs and requires precise specification for callee functions
due to the need for conjunction rule in the soundness proof

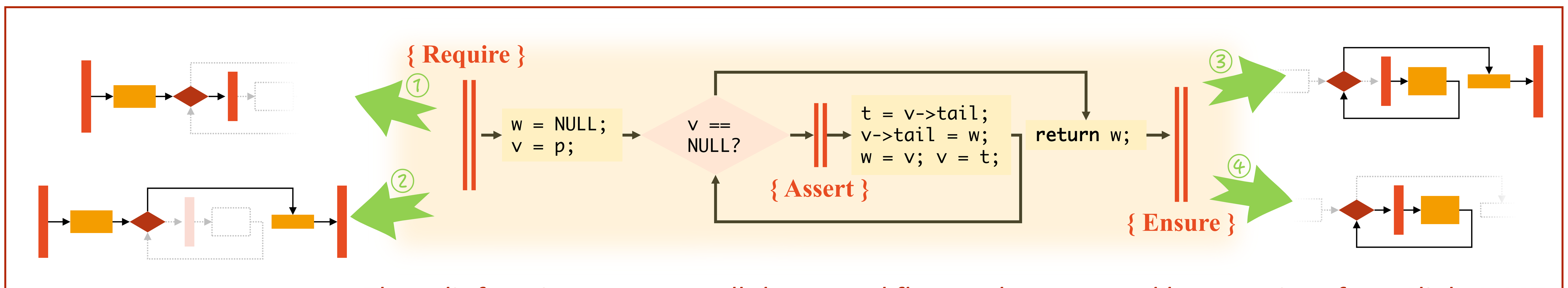
VST-A: to combine the benefits of interactive program verifiers as well as the readability of annotated programs

VST-A Workflow

```
struct list {
  unsigned head;
  struct list *tail;
};

struct list *reverse (struct list *p) {
/*@ With l,
    Require ll([[p]], l)
    Ensure ll([[ret]], rev(l)) */
  struct list *w, *t, *v;
  w = NULL; v = p;
  while (v) {
/*@ Assert ∃ l1 c x l2'.
      l = rev(l1) x l2' ∧ [[v]] ↦ (x,c)
      * ll([[w]], l1) * ll(c, l2') */
    t = v->tail; v->tail = w;
    w = v; v = t;
  }
  return w;
}
```

1. Users annotate C programs with function specifications and assertions as comments



The split function computes all the control flow paths separated by assertions for a ClightA AST

Parsing

$$
\begin{array}{rrcl}
\text{assertion}: & P & := & \cdots \\
\text{primary statement}: & c_p & := & e_1 := e_2 \\
 & & | & a := f(\vec{b}) \\
\text{ClightA statements}: & C & := & c_p \mid C_1; C_2 \\
 & & | & \text{if } (e)\ C_1 \text{ else } C_2 \\
 & & | & \text{loop } (C_2)\ C_1 \\
 & & | & \text{skip} \mid \text{break} \\
 & & | & \text{continue} \mid \text{return} \\
 & & | & \text{assert } P \\
 & & | & \text{ExGiven } x,\ \{P(x)\}\ C \\
\text{Annotation erasing}: & C\Downarrow & \in & \text{Clight statement}
\end{array}
$$

2. Annotated programs are parsed into **ClightA AST** definitions in Coq

Proved sound split function

split_res1.v
split_res2.v
split_res3.v
split_res4.v

Theorem. $\forall\, l\ l_1\ c\ x\ l_2'$,
$\{ l = \mathrm{rev}(l_1)\ x\ l_2' \land [\![v]\!] \mapsto (x, c) * \mathrm{ll}([\![w]\!], l_1) * \mathrm{ll}(c, l_2') \}$
t = v->tail; v->tail = w;
w = v; v = t; **assume** v;
$\{ \exists\, l_1\ c\ x\ l_2'.\ l = \mathrm{rev}(l_1)\ x\ l_2' \land [\![v]\!] \mapsto (x, c) * \mathrm{ll}([\![w]\!], l_1) * \mathrm{ll}(c, l_2') \}$
Proof. ...

3. A set of **straightline Hoare triples** are automatically computed and printed into separate Coq files

4. Users are left to prove residual proof goals that are not checked automatically.

* Areas marked by this color are user's verification obligations

Control-flow-based Split Function

We define an intermediate split result syntax to represent partial split results.

$$
\begin{array}{rrcl}
\text{Basic statement}: & c_b & := & c_p \mid \text{assume } e \\
\text{Assertion-free path}: & p_- & := & \vec{c_b} \\
\text{Head path}: & p_\dashv & := & \vec{c_b}\{P\} \\
\text{Tail path}: & p_\vdash & := & \{P\}\vec{c_b} \\
\text{Full path}: & p_\vdash\!\!\dashv & := & \{P_1\}\vec{c_b}\{P_2\} \mid \forall x.\ p_{\vdash\dashv}
\end{array}
$$

Figure. Syntax for Intermediate Split Results

The split function is defined by recursion on the ClightA AST and returns a record of (partial) paths.

Table. Definition of the Split Function (Selected Cases)

$\mathrm{split}(C) =$

Fields of split(C)	assert P	$C_1; C_2$	if (e) C_1 else C_2
Assertion-free paths with normal exits (p_-^{nor})	$\emptyset$	$p_-^{nor} \cdot q_-^{nor}$	$\{[\text{assume } e]\} \cdot p_-^{nor} \cup \{[\text{assume } \neg e]\} \cdot q_-^{nor}$
Assertion-free paths with return exits (p_-^{ret})	$\emptyset$	$p_-^{nor} \cdot q_-^{ret} \cup p_-^{ret}$	$\{[\text{assume } e]\} \cdot p_-^{ret} \cup \{[\text{assume } \neg e]\} \cdot q_-^{ret}$
Tail paths with normal exits ($p_\vdash^{nor}$)	$\{\{P\}[]\}$	$p_\vdash^{nor} \cdot q_-^{nor} \cup p_\vdash^{nor}$	$p_\vdash^{nor} \cup q_\vdash^{nor}$
Tail paths with return exits ($p_\vdash^{ret}$)	$\emptyset$	$p_\vdash^{ret} \cup q_\vdash^{ret} \cup p_\vdash^{nor} \cdot q_-^{ret}$	$p_\vdash^{ret} \cup q_\vdash^{ret}$
Head paths ($p_\dashv$)	$\{[]\{P\}\}$	$p_-^{nor} \cdot q_\dashv \cup p_\dashv$	$\{[\text{assume } e]\} \cdot p_\dashv \cup \{[\text{assume } \neg e]\} \cdot q_\dashv$
Full paths ($p_{\vdash\dashv}$)	$\emptyset$	$p_{\vdash\dashv} \cup q_{\vdash\dashv}$	$p_{\vdash\dashv} \cup q_{\vdash\dashv}$

assuming $\mathrm{split}(C_1) = \{p_-^{nor}, p_-^{ret}, p_\vdash^{nor}, p_\vdash^{ret}, p_\dashv, p_{\vdash\dashv}\}$ and $\mathrm{split}(C_2) = \{q_-^{nor}, q_-^{ret}, q_\vdash^{nor}, q_\vdash^{ret}, q_\dashv, q_{\vdash\dashv}\}$

Soundness of VST-A

Theorem (*Soundness*). For any ClightA program C and pre-/post-conditions P and Q, if $\mathrm{split}(C) = \{\emptyset, p_-^{ret}, \emptyset, p_\vdash^{ret}, p_\dashv, p_{\vdash\dashv}\}$ and all straightline Hoare triples below are provable:

$$
\left.\begin{array}{ll}
\text{(a) between internal assertions,} & p_{\vdash\dashv} \\
\text{(b) from } P \text{ to } Q, & \{P\}p_-\{Q\} \\
\text{(c) from } P \text{ to internal assertions,} & \{P\}p_\dashv \\
\text{(d) from internal assertions to } Q & p_\vdash^{ret}\{Q\}
\end{array}\right\}\ \textit{denoted as}\ \vdash_\forall \{P\}\ \mathrm{split}(C)\ \{Q\}
$$

then $C\Downarrow$ is functionally correct w.r.t. the specification, i.e. $\{P\}\ C\Downarrow\ \{Q\}$

Proof draft. by induction on C, case $[C = C_1; C_2]$ for example:

Condition: $\vdash_\forall \{P\}\ \mathrm{split}(C_1; C_2)\ \{Q\}$

Goal: find middle condition R, s.t. $\vdash_\forall \{P\}\ \mathrm{split}(C_1)\ \{?R\}$ and $\vdash_\forall \{?R\}\ \mathrm{split}(C_2)\ \{Q\}$

Solution: $R \triangleq$ conjunction of $\mathrm{split}(C_2)$'s weakest conditions

Proof of $\vdash_\forall \{P\}\ \mathrm{split}(C_1)\ \{?R\}$ requires the **conjunction rule**

$$
\textsc{Conj-Rule}\ \frac{\{P\}\, c\, \{Q_1\} \qquad \{P\}\, c\, \{Q_2\}}{\{P\}\, c\, \{Q_1 \land Q_2\}}
$$

Remark on Conjunction Rule

- The conjunction rule is natural in traditional Hoare logics for sequential programs, but some extensions to the logics (e.g. with ghost updates) will make this rule inadmissible.
- VST-A is currently based on a more restricted variant of VST program logic, that removes the ghost update operator, but retains all the other features.

Scan for the VST-A repository