Formal specification for RAVEN

1 Type System

We have the following built-in types:

1. Ref
2. Bool
3. Int
4. Set[T] for a type T.
5. List[T] for a type T.
6. Multiset[T] for a type T.

New types can be defined as follows:

1. Sum types.
2. Modules.

2 Grammar

$\mathit{prog} ::= \overline{\mathit{decl}}$

$\mathit{decl} ::= \mathit{proc\_decl} \mid \mathit{at\_proc\_decl} \mid \mathit{func\_decl} \mid \mathit{intf\_decl} \mid \mathit{module\_decl} \mid \mathit{module\_inst} \mid \mathit{pred\_decl}$

$\mathit{proc\_decl} ::=$ proc $\mathit{proc}(\overline{x : T})$
requires a
ensures a
$\{\mathit{stmt}\}$

$\mathit{at\_proc\_decl} ::=$ atomic proc $\mathit{proc}(\overline{x : T})$
requires a
ensures a
$\{\mathit{stmt}\}$

$\mathit{func\_decl} ::=$ function $\mathit{func}(\overline{x : T})$: T
requires a
ensures a
$\{\mathit{stmt}\}$

$\mathit{intf\_decl} ::=$ interface $\mathit{intf}[\overline{z : I}]$

$\mathit{module\_decl} ::=$ module $\mathit{mod}[\overline{z : I}] : \overline{\mathit{intf}}$ {e}

$\mathit{module\_inst} ::=$ module $\mathit{mod} = \mathit{mod}[\overline{z : I}]$

$\mathit{pred\_decl} ::=$ predicate $\mathit{pred}(\overline{x : T})$ {a}

$\mathit{stmt} ::= \mathit{stmt}; \mathit{stmt}$ | var x:T | x := e | x := new($\overline{f}$) | $\overline{x}$:= proc($\overline{e}$) | x.f := e | assert a | if (e) $\{\mathit{stmt}\}$ else $\{\mathit{stmt}\}$ | while (e)
invariant a $\{\mathit{stmt}\}$ | import intf | inhale/exhale/label/goto/fold/unfold

3 Semantics

Raven uses symbolic execution to do specification-based modular verification. Each method is verified with respect to its given specification, by starting from a symbolic state which represents the precondition and symbolically executing each statement to ensure that the postconditions hold at the final state.

Raven uses an SMT solver as the back end to discharge proof obligations. Like Smallfoot and Viper, Raven splits the symbolic state into *path conditions* which store pure assertions expressed in first order logic, and a *symbolic heap* which assigns symbolic values to locations.

The symbolic state consists of:

- A *store* γ which maps local variables to their symbolic values.
- A *path condition stack* π that stores triples $(\mathit{Id}, V, \mathit{Set}[V])$ of a unique scope identifier, a branch condition, and a set of path conditions.
- A symbolic heap h, which is a multiset of heap chunks of the shape $id(\overline{v}; \overline{w})$. For instance for a field f, $f(r; v, p)$ denotes that $r.f$ has the symbolic value v and permission p.
- A *ghost symbolic heap* g, which stores the values of the ghost locations.
- A *labelled heaps* map lbh that maps identifiers to symbolic heaps.