IMP

MODULE SYMBOLIC-ARRAY-SYNTAX

SYNTAX $KResult$ ::= `select` ($Array$, Int) [smtlib(select)]

END MODULE

MODULE SYMBOLIC-ARRAY

END MODULE

> This is the symbolic semantics of IMP enriched with arrays and Hoare logic. The semantics, gets as input an IMP program, annotated with pre and post conditions and invariants.

MODULE IMP-SYNTAX

SYNTAX $AExp$ ::= Int
| Id
| $AExp[AExp]$ [strict]
| $AExp$ * $AExp$ [strict]
| $AExp$ / $AExp$ [strict]
| $AExp$ + $AExp$ [strict]
| $AExp$ - $AExp$ [strict]
| ($AExp$) [bracket]

SYNTAX $BExp$ ::= $Bool$
| $AExp \leq AExp$ [seqstrict]
| $AExp$ == $AExp$ [strict]
| ! $BExp$ [strict]
| $BExp$ && $BExp$ [strict(1)]
| ($BExp$) [bracket]

SYNTAX $Block$ ::= {}
| {$Stmt$}

SYNTAX $Stmt$ ::= $Block$
| $AExp$ = $AExp$; [strict(2)]
| `if` ($BExp$)$Block$ `else` $Block$ [strict(1)]
| `while` ($BExp$)$Block$
| $Stmt$ $Stmt$

SYNTAX Pgm ::= `int` $AExps$; $Stmt$

SYNTAX $AExps$ ::= $List\{AExp, \text{“,”}\}$

SYNTAX Ids ::= $List\{Id, \text{“,”}\}$

SYNTAX $Stmt$ ::= `while` ($BExp$) `invariant` : $Assert$ $Block$

SYNTAX $Assert$ ::= $BExp$
| `not` $Assert$ [strict]
| $Assert$ `and` $Assert$ [strict]
| $Assert$ `or` $Assert$ [strict]
| $Assert$ `implies` $Assert$ [strict]
| `forall` Ids($Assert$) [strict(2)]
| `exists` Ids($Assert$) [strict(2)]

SYNTAX Pre ::= `pre` : $Assert$

SYNTAX $Post$ ::= `post` : $Assert$

SYNTAX $Program$ ::= `int` $AExps$; Pre $Post$ $Stmt$

SYNTAX Val ::= Int
| $Bool$
| $Array$
| `array` (Int, Int)
| `loc` ($AExp$)

SYNTAX $AExp$ ::= Val

END MODULE

MODULE IMP

SYNTAX $KResult$ ::= Val

CONFIGURATION:

T: [k: $\$PGM{:}Program$ | env: $\bullet_{Map}$ | store: $\bullet_{Map}$ | condition: true]

> IMP concrete semantics

RULE k: $\frac{X{:}Id}{V}$ env: $X \mapsto L$ store: $L{:}Int \mapsto V{:}Val$

RULE k: $\frac{\texttt{array}\ (L{:}Int, Size{:}Int)[I{:}Int]}{\texttt{lookup}\ (L +_{Int} I)}$ store: $L \mapsto$ —:#Int

RULE k: $\frac{\texttt{array}\ (L{:}Int, Size{:}Int)[I{:}Int]}{\texttt{select}\ (A, I)}$ store: $L \mapsto A{:}Array$

RULE $\frac{I1{:}Int + I2{:}Int}{I1 +_{Int} I2}$

RULE $\frac{I1{:}Int - I2{:}Int}{I1 -_{Int} I2}$

RULE $\frac{I1{:}Int * I2{:}Int}{I1 *_{Int} I2}$

RULE $\frac{I1{:}Int \leq I2{:}Int}{I1 \leq_{Int} I2}$

RULE $\frac{I1{:}Int == I2{:}Int}{I1 ==_{Int} I2}$

RULE $\frac{!\ T{:}Bool}{\neg_{Bool} T}$

RULE $\frac{\{\}}{\bullet_K}$ [structural]

RULE $\frac{\{S\}}{S}$ [structural]

SYNTAX $AExp$ ::= `lvalue` (K)

RULE k: `lvalue` ($\frac{X{:}Id}{\texttt{loc}\ (L)}$) env: $X \mapsto L{:}Int$ [structural]

SYNTAX K ::= `lookup` (Int)

RULE k: $\frac{\texttt{lookup}\ (L)}{V}$ store: $L \mapsto V{:}Val$ [lookup]

CONTEXT $\frac{\square}{\texttt{lvalue}\ (\square)}$ = — ;

CONTEXT `lvalue` (—[□])

CONTEXT `lvalue` (□[—])

RULE k: $\frac{\texttt{loc}\ (L{:}Int) = I{:}Int\ ;}{\bullet_K}$ store: $L \mapsto \frac{—}{I}$

RULE k: $\frac{\texttt{loc}\ (\ \texttt{array}\ (L{:}Int, Size{:}Int)[I{:}Int]) = V{:}Val\ ;}{\bullet_K}$ store: $L \mapsto \frac{A{:}Array}{\texttt{store}\ (A, I, V)}$

RULE k: $\frac{\texttt{lvalue}\ (\ \texttt{array}\ (L{:}Int, Size{:}Int)[I{:}Int])}{\texttt{loc}\ (L +_{Int} I)}$ store: $L \mapsto$ —:Int

RULE $\frac{S1\ \ S2}{S1 \curvearrowright S2}$ [structural]

RULE $\frac{\texttt{if}\ (\text{true}) S\ \texttt{else}\ —}{S}$

RULE $\frac{\texttt{if}\ (\text{false}) —\ \texttt{else}\ S}{S}$

RULE $\frac{\texttt{while}\ (B) S}{\texttt{if}\ (B)\{S\ \ \texttt{while}\ (B) S\}\ \texttt{else}\ \{\}}$ [structural]

RULE k: `int` $\frac{X{:}Id\ ,\ Xs{:}AExps}{Xs}$; —:Pre —:Post —:Stmt env: $\rho{:}Map\ \frac{\bullet_{Map}}{X \mapsto L}$ store: $\frac{\bullet_{Map}}{(L \mapsto 0)}$ requires $\neg_{Bool}(X\ \texttt{in}\ \texttt{keys}\ (\rho)) \wedge_{Bool}\ \texttt{fresh}\ (L{:}Int)$

RULE k: `int` $\frac{X{:}Id[S{:}\#Int]\ ,\ Xs{:}Ids}{Xs}$; —:Pre —:Post $St{:}Stmt$ env: $\rho{:}Map\ \frac{\bullet_{Map}}{X \mapsto L}$ store: $\frac{\bullet_{Map}}{L \mapsto\ \texttt{array}\ (AL, S)\ \ AL \mapsto \#symArray(X)}$ requires $\neg_{Bool}(X\ \texttt{in}\ \texttt{keys}\ (\rho)) \wedge_{Bool}\ \texttt{fresh}\ (L{:}Int) \wedge_{Bool}\ \texttt{fresh}\ (AL{:}Int)$

RULE $\frac{\texttt{int}\ \bullet_{AExps}\ ;\ P{:}Pre\ \ P'{:}Post\ \ S{:}Stmt}{P \curvearrowright P' \curvearrowright S}$ [structural]

> IMP symbolic semantics

RULE T: [k: $\frac{I1{:}Int\ /\ I2{:}Int}{I1 \div_{Int} I2}$ condition: $\frac{Phi{:}Bool}{Phi \wedge_{Bool} I2 =/=_{Int} 0}$] requires `checkSat` $(Phi \wedge_{Bool} (I2 =/=_{Int} 0)) =_K$ "sat" [transition]

RULE T: [k: $\frac{B1{:}Bool\ \&\&\ B2{:}BExp}{B2}$ condition: $\frac{Phi{:}Bool}{Phi \wedge_{Bool} B1\ \text{==Bool true}}$] requires `checkSat` $(Phi \wedge_{Bool} B1) =_K$ "sat" [transition]

RULE T: [k: $\frac{B1{:}Bool\ \&\&\ B2{:}Bool}{\text{false}}$ condition: $\frac{Phi{:}Bool}{Phi \wedge_{Bool} \neg_{Bool} B1}$] requires `checkSat` $(Phi \wedge_{Bool} \neg_{Bool} B1) =_K$ "sat" [transition]

RULE T: [k: $\frac{\texttt{if}\ (B{:}Bool) S{:}Block\ \texttt{else}\ —}{S}$ condition: $\frac{Phi{:}Bool}{Phi \wedge_{Bool} B}$] requires `checkSat` $(Phi \wedge_{Bool} B) =_K$ "sat" [transition]

RULE T: [k: $\frac{\texttt{if}\ (B{:}Bool) —\ \texttt{else}\ S{:}Block}{S}$ condition: $\frac{Phi{:}Bool}{Phi \wedge_{Bool} \neg_{Bool} B}$] requires `checkSat` $(Phi \wedge_{Bool} \neg_{Bool} B) =_K$ "sat" [transition]

> Hoare triples

RULE k: $\frac{(\ \texttt{pre}\ :\ Psi{:}Assert \curvearrowright\ \texttt{post}\ :\ Psi'{:}Assert \curvearrowright S{:}Stmt)}{\texttt{assume}\ (Psi) \curvearrowright S \curvearrowright\ \texttt{match}\ (\ \texttt{A2M}\ (Psi'))}$

> Assume

SYNTAX K ::= `assume` (K)
| `assumeStrict` (K) [strict]

RULE $\frac{\texttt{assume}\ (Psi{:}K)}{\texttt{assumeStrict}\ (\ \texttt{A2M}\ (Psi))}$

RULE k: $\frac{\texttt{assumeStrict}\ (\ \texttt{m2k}\ (E{:}Map, Phi{:}K))}{\bullet_K}$ store: $\frac{—}{E}$ condition: $\frac{—}{Phi}$

> Match

SYNTAX K ::= `match` (K) [strict]

RULE k: $\frac{\texttt{match}\ (\ \texttt{m2k}\ (Ea{:}Map, Phia{:}K))}{\bullet_K}$ store: $Ec{:}Map$ condition: $Phic{:}Bool$ requires `checkSat` $(Phic \wedge_{Bool} (\neg_{Bool} Phia) \wedge_{Bool}\ \texttt{mapLeftEq}\ (Ea, Ec)) =_K$ "unsat"

RULE k: $\frac{\texttt{match}\ (\ \texttt{m2k}\ (Ea{:}Map, Phia{:}K))}{\texttt{SMTLibQuery}\ (Phic \wedge_{Bool} (\neg_{Bool} Phia) \wedge_{Bool}\ \texttt{mapLeftEq}\ (Ea, Ec))}$ store: $Ec{:}Map$ condition: $Phic{:}Bool$ requires `checkSat` $(Phic \wedge_{Bool} (\neg_{Bool} Phia) \wedge_{Bool}\ \texttt{mapLeftEq}\ (Ea, Ec)) =_K$ "sat"

> While invariant

RULE k: $\frac{\texttt{while}\ \ (B{:}BExp)\ \texttt{invariant}\ :\ Psi{:}Assert\ \ S{:}Block \curvearrowright K}{\texttt{assume}\ (Psi\ \texttt{and}\ B) \curvearrowright S \curvearrowright\ \texttt{match}\ (\ \texttt{A2M}\ (Psi))}$ [transition]

RULE k: $\frac{\texttt{while}\ \ (B{:}BExp)\ \texttt{invariant}\ :\ Psi{:}Assert\ \ —{:}\text{Block}}{\texttt{match}\ (\ \texttt{A2M}\ (Psi)) \curvearrowright\ \texttt{assume}\ (Psi\ \texttt{and not}\ B)}$ [transition]

> A2M

SYNTAX K ::= `A2M` (K)

RULE k: $\frac{\texttt{A2M}\ (Psi)}{\texttt{generateFresh}\ (E) \curvearrowright\ \texttt{evalAssert}\ (Psi) \curvearrowright\ \texttt{restore}\ (E, Phi)}$ store: $E{:}Map$ condition: $Phi{:}Bool$

> Generate fresh map

SYNTAX K ::= `generateFresh` (Map)

RULE k: $\frac{\texttt{generateFresh}\ (L{:}Int \mapsto V{:}Int\ \ E{:}Map)}{\texttt{generateFresh}\ (E)}$ store: $L \mapsto \frac{V}{V'}$ requires `fresh` $(V'{:}Int)$

RULE k: $\frac{\texttt{generateFresh}\ (\bullet_{Map})}{\bullet_K}$

RULE k: $\frac{\texttt{generateFresh}\ (L{:}Int \mapsto\ \texttt{array}\ (AL{:}Int, Size{:}Int)\ \ E{:}Map)}{\texttt{generateFresh}\ (E)}$

RULE k: $\frac{\texttt{generateFresh}\ (L{:}Int \mapsto A{:}Array\ \ E{:}Map)}{\texttt{generateFresh}\ (E)}$

> Assert evaluation

SYNTAX K ::= `evalAssert` (K) [strict]

SYNTAX $KResult$ ::= `m2k` (Map, K)

RULE k: $\frac{\texttt{evalAssert}\ (Phi{:}K) \curvearrowright\ \texttt{restore}\ (E'{:}Map, Phi'{:}K)}{\texttt{m2k}\ (E, Phi)}$ store: $\frac{E{:}Map}{E'}$ condition: $\frac{—}{Phi'}$

> Restore

SYNTAX K ::= `restore` (Map, $Bool$)

> Assertions

RULE $\frac{B1{:}Bool\ \texttt{and}\ B2{:}Bool}{B1 \wedge_{Bool} B2}$

RULE $\frac{B1{:}Bool\ \texttt{implies}\ B2{:}Bool}{B1\ \texttt{impliesBool}\ B2}$

RULE $\frac{B1{:}Bool\ \texttt{or}\ B2{:}Bool}{B1 \vee_{Bool} B2}$

RULE $\frac{\texttt{not}\ B{:}Bool}{\neg_{Bool} B}$

RULE $\frac{\texttt{forall}\ Is{:}Ids(B{:}Bool)}{\texttt{forall toSet}\ (Is)\ .\ B}$

RULE $\frac{\texttt{exists}\ Is{:}Ids(B{:}Bool)}{\texttt{exists toSet}\ (Is)\ .\ B}$

SYNTAX Set ::= `toSet` (Ids) [function]

RULE $\frac{\texttt{toSet}\ (\bullet_{Ids})}{\bullet_{Set}}$

RULE $\frac{\texttt{toSet}\ (X{:}Id\ ,\ Is{:}Ids)}{\#symInt(X)\ \ \texttt{toSet}\ (Is)}$

> Utils

SYNTAX $Bool$ ::= `mapLeftEq` (Map, Map) [function]

RULE $\frac{\texttt{mapLeftEq}\ (X{:}Int \mapsto V1{:}Int\ \ Rest{:}Map, Left{:}Map\ \ X \mapsto V2\ \ Right{:}Map)}{V1 ==_{Int} V2 \wedge_{Bool}\ \texttt{mapLeftEq}\ (Rest, Left\ \ Right)}$

RULE $\frac{\texttt{mapLeftEq}\ (X{:}Int \mapsto\ \texttt{array}\ (AL{:}Int, Sz{:}Int)\ \ Rest{:}Map, Left{:}Map\ \ X \mapsto\ \texttt{array}\ (AL, Sz)\ \ Right{:}Map)}{\texttt{mapLeftEq}\ (Rest, Left\ \ Right)}$

RULE $\frac{\texttt{mapLeftEq}\ (X{:}Int \mapsto A{:}Array\ \ Rest{:}Map, Left{:}Map\ \ X \mapsto A\ \ Right{:}Map)}{\texttt{mapLeftEq}\ (Rest, Left\ \ Right)}$

RULE $\frac{\texttt{mapLeftEq}\ (\bullet_{Map}, —)}{\text{true}}$

RULE $\frac{\texttt{mapLeftEq}\ (X{:}Int \mapsto —{:}\text{Int}\ \ —{:}\text{Map}, Right{:}Map)}{\text{false}}$ requires $\neg_{Bool}(X\ \texttt{in}\ \texttt{keys}\ (Right))$

RULE $\frac{\texttt{mapLeftEq}\ (M{:}Map, \bullet_{Map})}{\text{false}}$ requires $\neg_{Bool}(\ \texttt{isEmptySet}\ (\ \texttt{keys}\ (M)))$

> Compiler issues

RULE $\frac{isSymbolicInt(\ \texttt{select}\ (A{:}Array, I{:}Int))}{\text{true}}$ [anywhere]

RULE $\frac{isInt(\ \texttt{select}\ (A{:}Array, I{:}Int))}{\text{true}}$ [anywhere]

RULE $\frac{\texttt{lvalue}\ (\ \texttt{array}\ (I{:}Int, S{:}Int)[V{:}Int])}{\texttt{lvalue}\ (\ \texttt{loc}\ (\ \texttt{array}\ (I{:}Int, S{:}Int)[V{:}Int]))}$

RULE $\frac{\texttt{K2Sort}\ (—{:}\text{Array})}{\text{"(Array Int Int)"}}$

RULE k: $\frac{X{:}Id}{\#symInt(X)}$ env: ρ requires $\neg_{Bool}(X\ \texttt{in}\ \texttt{keys}\ (\rho))$

END MODULE