

# IMP

MODULE SYMBOLIC-ARRAY-SYNTAX

SYNTAX  $KResult ::= \text{select } (Array, Int) \text{ [symbolicselect]}$

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  $AExp ::= Int$   
 $| Id$   
 $| AExp[AExp] \text{ [strict]}$   
 $| AExp * AExp \text{ [strict]}$   
 $| AExp / AExp \text{ [strict]}$   
 $| AExp + AExp \text{ [strict]}$   
 $| AExp - AExp \text{ [strict]}$   
 $| (AExp) \text{ [bracket]}$

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

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

SYNTAX  $Smt ::= Block$   
 $| AExp = AExp : \text{[strict(2)]}$   
 $| \text{if } (BExp) Block \text{ else } Block \text{ [strict(1)]}$   
 $| \text{while } (BExp) Block$   
 $| Smt \ Smt$

SYNTAX  $Pgm ::= \text{int } AExps ; Smt$

SYNTAX  $AExps ::= List(AExp, ",")$

SYNTAX  $Ids ::= List(Id, ",")$

SYNTAX  $Smt ::= \text{while } (BExp) \text{ invariant} : \text{Assert } Block$

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

SYNTAX  $Pre ::= \text{pre} : Assert$

SYNTAX  $Post ::= \text{post} : Assert$

SYNTAX  $Program ::= \text{int } AExps ; Pre \ Post \ Smt$

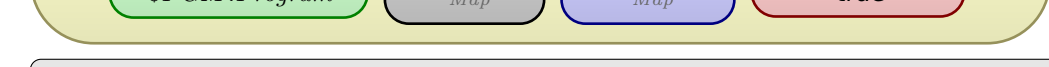
SYNTAX  $Val ::= Int$   
 $| Bool$   
 $| Array$   
 $| \text{array } (Int, Int)$   
 $| \text{loc } (AExp)$

END MODULE

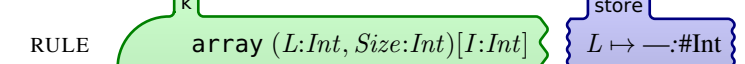
MODULE IMP

SYNTAX  $KResult ::= Val$

CONFIGURATION:



IMP concrete semantics



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

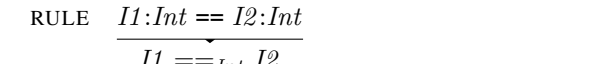
RULE  $\frac{I1: Int == I2: Int}{I1 ==_{Int} I2}$

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

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

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

SYNTAX  $AExp ::= \text{lvalue } (K)$



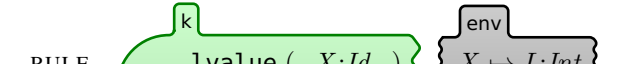
SYNTAX  $K ::= \text{lookup } (Int)$



CONTEXT  $\frac{}{\square} = -;$

CONTEXT  $\text{lvalue } (-[\square])$

CONTEXT  $\text{lvalue } (\square[-])$

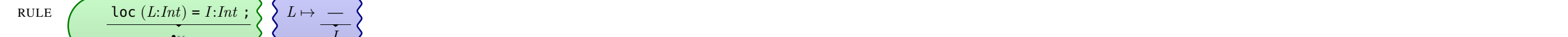
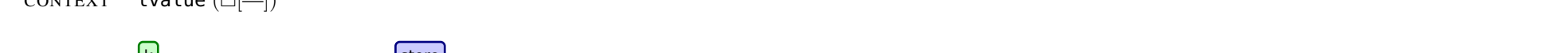


RULE  $\frac{S1 \ S2}{S1 \sim S2}$  [structural]

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

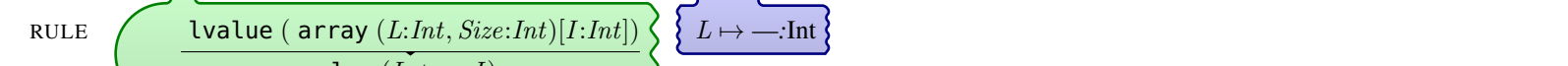
RULE  $\frac{\text{if } (\text{false}) - \text{ else } S}{S}$

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

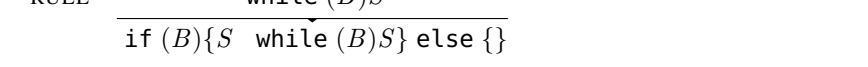


RULE  $\frac{\text{int } \star_{(AExp)} ; P:Pre \ P':Post \ S:Smt}{P \sim P' \sim S}$  [structural]

IMP symbolic semantics



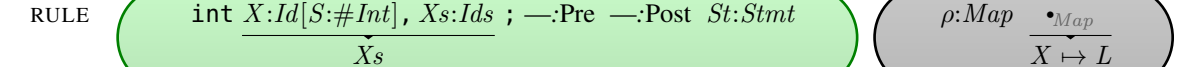
Hoare triples



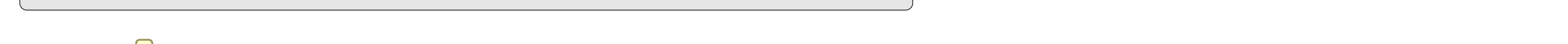
Assume

SYNTAX  $K ::= \text{assume } (K)$   
 $| \text{assumeStrict } (K) \text{ [strict]}$

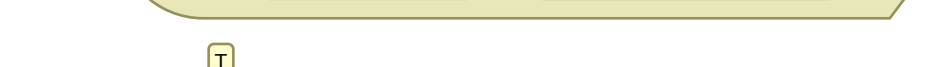
RULE  $\frac{\text{assume } (Psi:K)}{\text{assumeStrict } (A2M (Psi))}$



Match

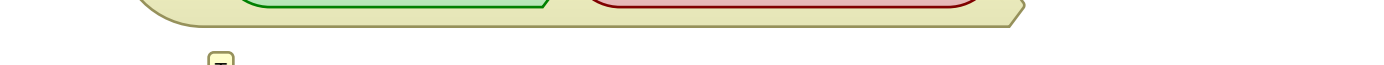


While invariant



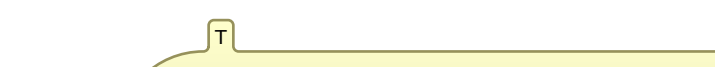
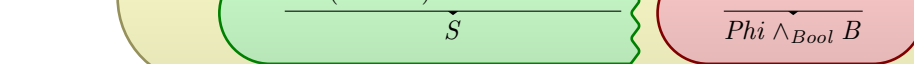
A2M

SYNTAX  $K ::= A2M (K)$



Generate fresh map

SYNTAX  $K ::= \text{generateFresh } (Map)$



Assert evaluation

SYNTAX  $K ::= \text{evalAssert } (K) \text{ [strict]}$

SYNTAX  $KResult ::= \text{m2k } (Map, K)$



Restore

SYNTAX  $K ::= \text{restore } (Map, Bool)$

Assertions

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

RULE  $\frac{B1:Bool \text{ implies } B2:Bool}{B1 \text{ implies}_{Bool} B2}$

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

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

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

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

SYNTAX  $Ser ::= \text{toSet } (Ids) \text{ [function]}$

RULE  $\frac{\text{toSet } (\star_{(Is)})}{\sim_{Set}}$

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

Utils

SYNTAX  $Bool ::= \text{mapLeftEq } (Map, Map) \text{ [function]}$

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

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

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

RULE  $\frac{\text{mapLeftEq } (\star_{(Int)} \mapsto -)}{\text{true}}$

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

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

Compiler issues

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

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

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

RULE  $\frac{\text{K2Sort } (-;Array)}{"(Array \text{Int } \text{Int})"}$

RULE  $\frac{\text{X:Id } \#symInt(X)}{\#symInt(X)}$  requires  $\neg_{Bool} (X \text{ in keys } (\rho))$

END MODULE