

IMP

MODULE SYMBOLIC-ARRAY-SYNTAX

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

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

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

SYNTAX $Block ::= \{\}$
 $\quad \mid \{ Stmt \}$

SYNTAX $Stmt ::= Block$
 $\quad \mid AExp \leftarrow AExp : \text{strict}(2)$
 $\quad \mid \text{if } (BExp) Block \text{ else } Block \text{ [strict(1)]}$
 $\quad \mid \text{while } (BExp) Block$
 $\quad \mid Stmt Stmt$

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

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

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

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

SYNTAX $Assert ::= BExp$
 $\quad \mid \neg \text{Assert} \text{ [strict]}$
 $\quad \mid \text{Assert and Assert} \text{ [strict]}$
 $\quad \mid \text{Assert or Assert} \text{ [strict]}$
 $\quad \mid \text{Assert implies Assert} \text{ [strict]}$
 $\quad \mid \text{forall } Ids(\text{Assert}) \text{ [strict(2)]}$
 $\quad \mid \text{exists } Ids(\text{Assert}) \text{ [strict(2)]}$

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

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

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

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

END MODULE

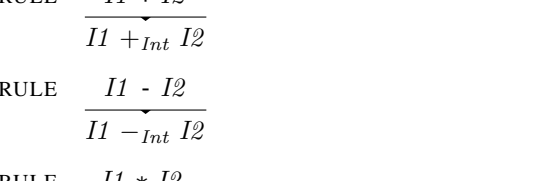
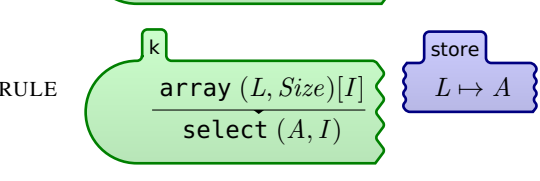
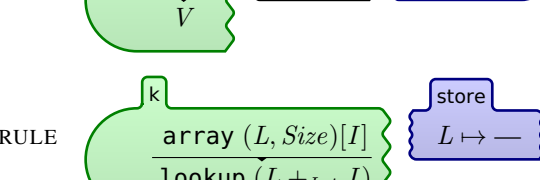
MODULE IMP

SYNTAX $KResult ::= Val$

CONFIGURATION:



IMP concrete semantics



RULE $\frac{I1 + I2}{I1 +_{int} I2}$

RULE $\frac{I1 - I2}{I1 -_{int} I2}$

RULE $\frac{I1 * I2}{I1 *_{int} I2}$

RULE $\frac{I1 \leq I2}{I1 \leq_{int} I2}$

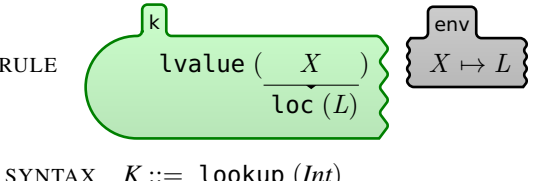
RULE $\frac{I1 == I2}{I1 ==_{int} I2}$

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

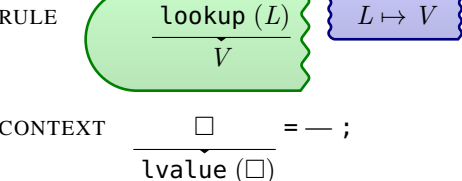
RULE $\frac{\{\}}{*_{\epsilon}}$

RULE $\frac{\{S\}}{S}$

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



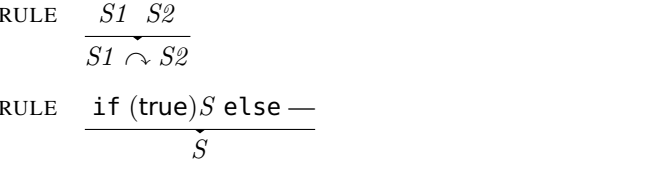
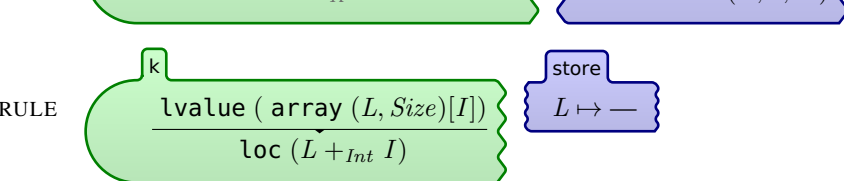
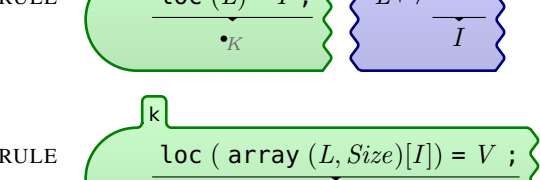
SYNTAX $K ::= \text{lookup } (Int)$



CONTEXT $\frac{}{\text{lvalue}(\Box) = \text{—}} ;$

CONTEXT $\text{lvalue}(\text{—}[\Box])$

CONTEXT $\text{lvalue}(\Box[\text{—}])$

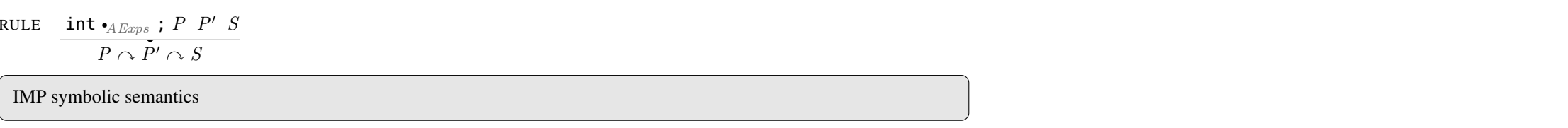
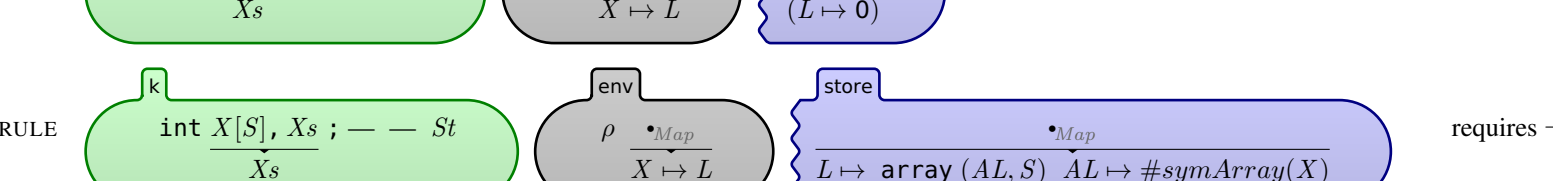


RULE $\frac{S1 \ S2}{S1 \wedge S2}$

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

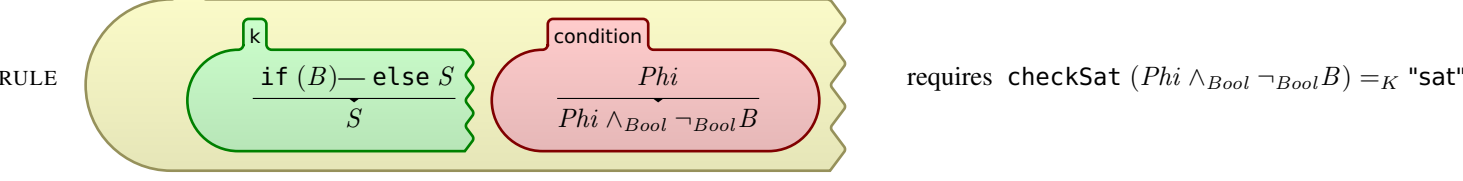
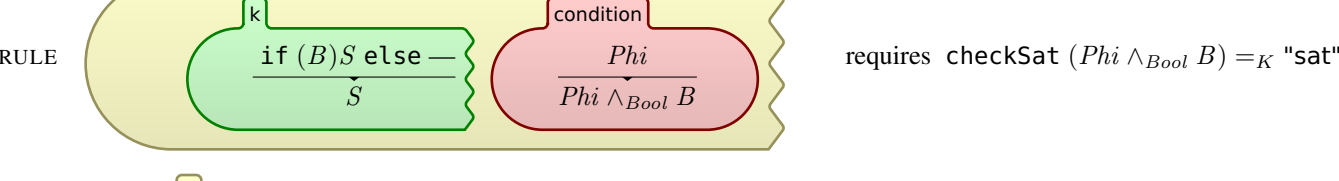
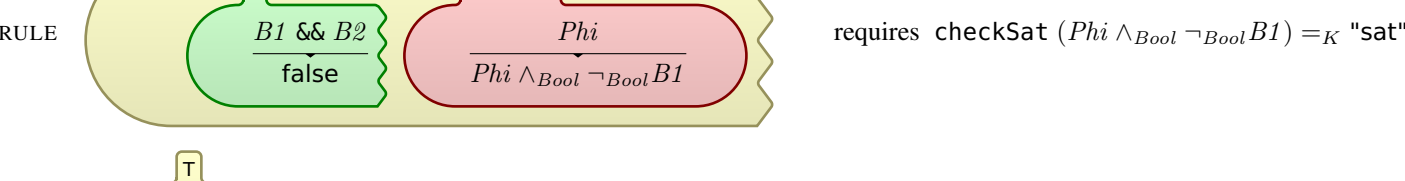
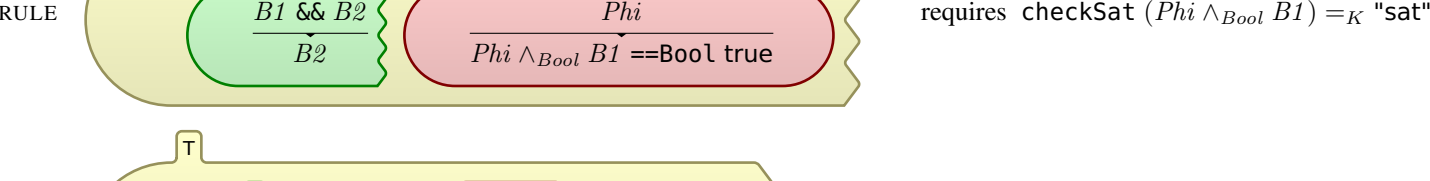
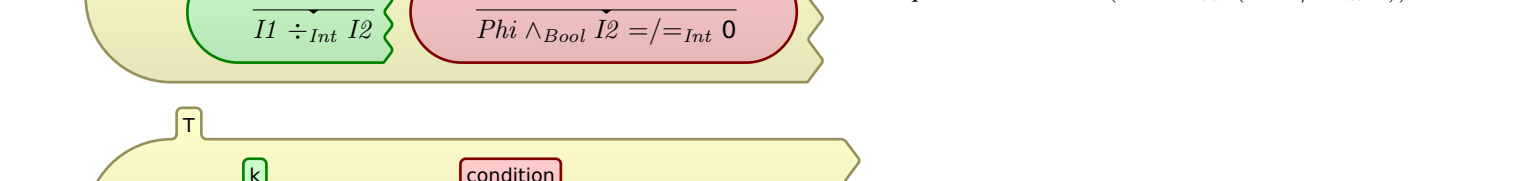
RULE $\frac{\text{if } (\text{false}) \text{—} \text{ else } S}{S}$

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

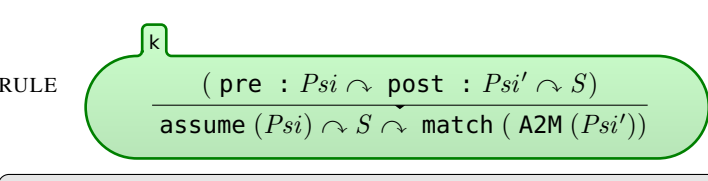


RULE $\frac{\text{int } *_{Eexp} ; P \ P' \ S}{P \wedge P' \wedge S}$

IMP symbolic semantics



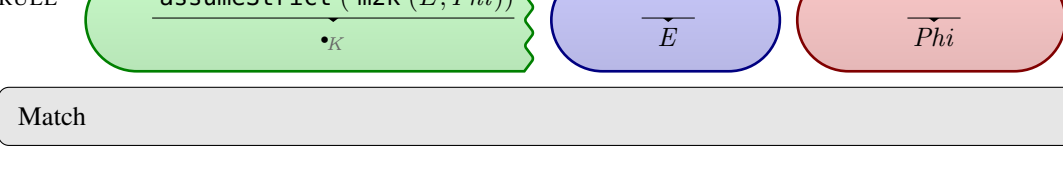
Hoare triples



Assume

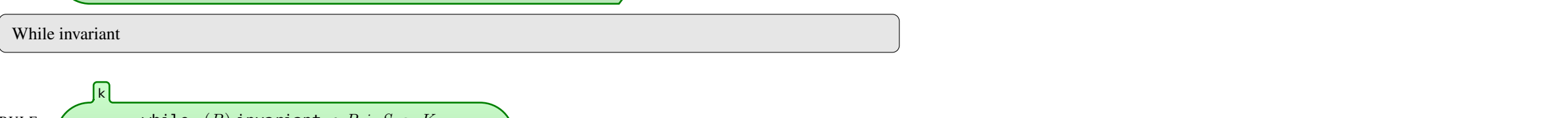
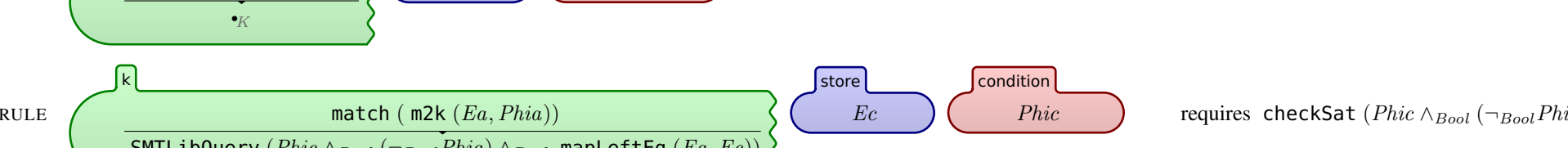
SYNTAX $K ::= \text{assume } (K)$
 $\quad \mid \text{assumeStrict } (K) \text{ [strict]}$

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

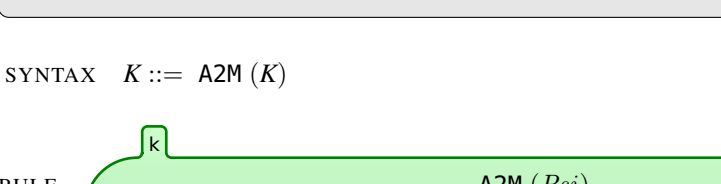
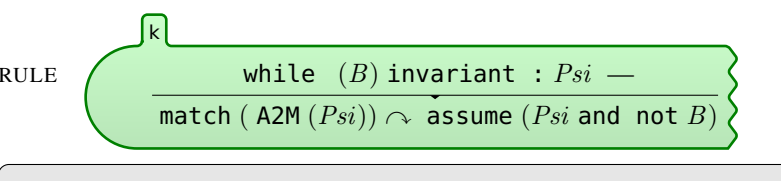


Match

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

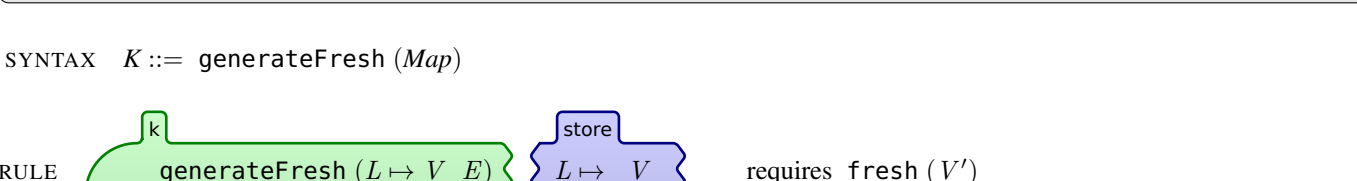


While invariant



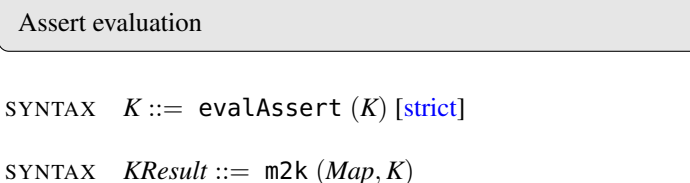
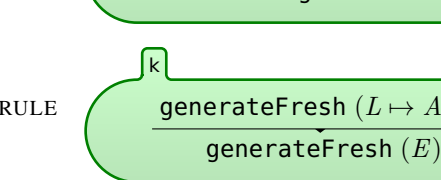
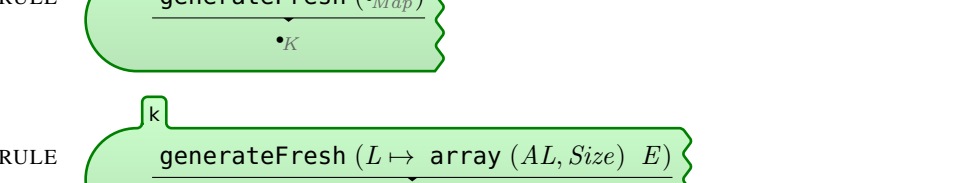
A2M

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 $B1 \text{ and } B2$

RULE $B1 \wedge_{Bool} B2$

RULE $B1 \text{ implies } B2$

RULE $B1 \text{ impliesBool } B2$

RULE $B1 \text{ or } B2$

RULE $B1 \vee_{Bool} B2$

RULE $\text{not } B$

RULE $\neg_{Bool} B$

RULE $\text{forall } Is(B)$

RULE $\text{forall toSet } (Is) . B$

RULE $\text{exists } Is(B)$

RULE $\text{exists toSet } (Is) . B$

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

RULE $\text{toSet } (*_{Is})$

RULE $\text{toSet } (X, Is)$

RULE $\#symInt(X) \text{ toSet } (Is)$

Utils

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

RULE $\text{mapLeftEq } (X \mapsto V1 \text{ Rest, Left } X \mapsto V2 \text{ Right})$

RULE $V1 ==_{int} V2 \wedge_{Bool} \text{mapLeftEq } (Rest, Left \text{ Right})$

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

RULE $\text{mapLeftEq } (X \mapsto A \text{ Rest, Left } X \mapsto A \text{ Right})$

RULE $\text{mapLeftEq } (Rest, Left \text{ Right})$

RULE $\text{mapLeftEq } (*_{lookup}, \text{—})$

RULE $\text{mapLeftEq } (X \mapsto \text{—}, \text{Right})$ requires $\neg_{Bool}(X \text{ in keys } (Right))$

RULE $\text{mapLeftEq } (M, *_{lookup})$ requires $\neg_{Bool}(\text{isEmptySet } (\text{keys } (M)))$

Compiler issues

RULE $\text{isSymbolicInt}(\text{select } (A, I))$

RULE $\text{isInt}(\text{select } (A, I))$

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

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

RULE $K2\text{Sort } (\text{—})$

RULE $"(Array \text{ Int Int})"$

END MODULE