IMP

This is the symbolic semantics of IMP. It contains the normal semantics if IMP and the rules for symbolic execution, as described in the technical report available here: https://fmse.info.uaic.ro/publications/156/ .

MODULE IMP-SYNTAX

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

SYNTAX *BExp* ::= *Bool*
| *AExp* ≤ *AExp* [seqstrict]
| `!` *BExp* [strict]
| *BExp* && *BExp* [strict(1)]
| (*BExp*) [bracket]

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

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

SYNTAX *Pgm* ::= `int` *Ids* `;` *Stmt*

SYNTAX *Ids* ::= *List*{*Id*,“,”}

Programs are $\mathbb{K}$ configurations.

SYNTAX *Stmt* ::= `#ps` (*Bag*)

Assertions syntax

SYNTAX *Stmt* ::= `assert` (*BExp*) `;` [strict]

$\mathbb{K}$ compiler issues

SYNTAX *Dummy* ::= `symInt` [dummySymInt]

SYNTAX *Int* ::= `#symInt` (*Id*) [onlyLabel, klabel(#symInt)]

SYNTAX *Id* ::= *Token*{”*a*”}
| *Token*{”*b*”}
| *Token*{”*c*”}
| *Token*{”*sum*”}
| *Token*{”*i*”}
| *Token*{”*n*”}
| *Token*{”*x*”}
| *Token*{”*min*”}

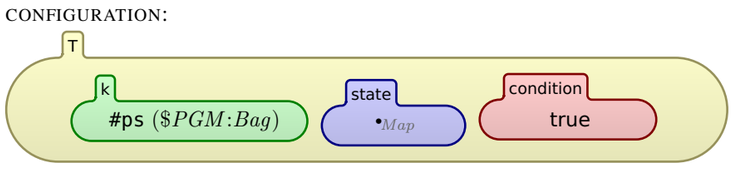
END MODULE

MODULE IMP

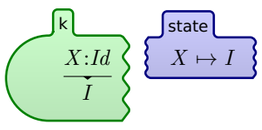
SYNTAX *KResult* ::= *Int*
| *Bool*

IMP configuration is enriched with cell `condition`.

CONFIGURATION:



The concrete semantics of IMP which remains unchanged for symbolic execution

RULE 

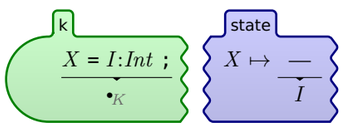
RULE $\dfrac{I1{:}Int \texttt{ + } I2{:}Int}{I1 +_{Int} I2}$

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

RULE $\dfrac{\texttt{!}\ T{:}Bool}{\neg_{Bool} T}$

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

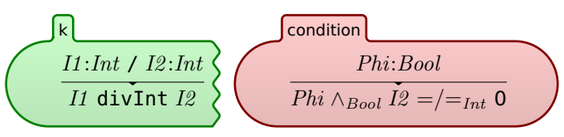
RULE $\dfrac{\{S\}}{S}$ [structural]

RULE 

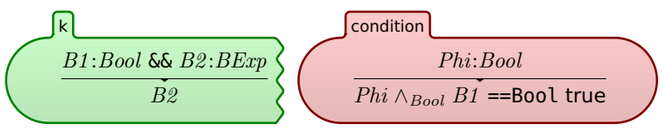
RULE $\dfrac{S1 \quad S2}{S1 \curvearrowright S2}$ [structural]

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

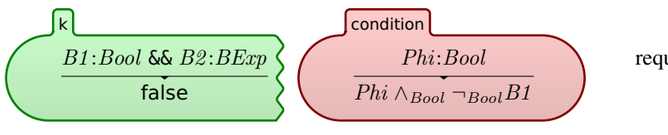
IMP symbolic semantics

RULE 

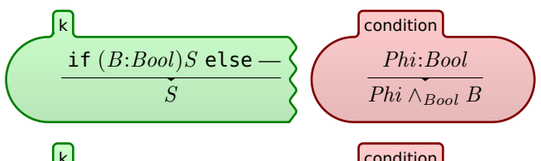
requires `checkSat` ($Phi \wedge_{Bool} (I2 =/=_{Int} 0)) \neq_K$ "unsat" [transition]

RULE 

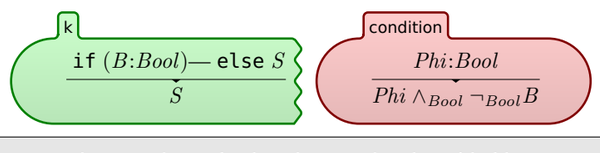
requires `checkSat` ($Phi \wedge_{Bool} B1) \neq_K$ "unsat" [transition]

RULE 

requires `checkSat` ($Phi \wedge_{Bool} \neg_{Bool} B1) \neq_K$ "unsat" [transition]

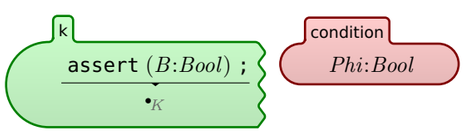
RULE 

requires `checkSat` ($Phi \wedge_{Bool} B) \neq_K$ "unsat" [transition]

RULE 

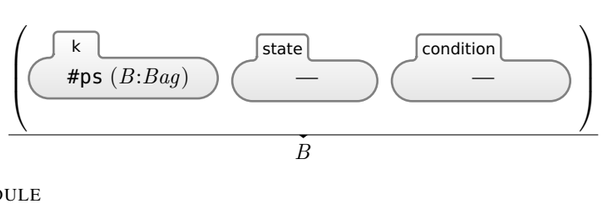
requires `checkSat` ($Phi \wedge_{Bool} \neg_{Bool} B) \neq_K$ "unsat" [transition]

Assert semantics: remain stuck when the assertion doesn’ hold.

RULE 

requires `checkSat` ($Phi \wedge_{Bool} B) =_K$ "sat"

Load PGM.

RULE 

[structural]

END MODULE