MODULE KWALA-SYNTAX

Syntax

...

SYNTAX *Rhs* ::= `object`
| `null`
SYNTAX *Variable* ::= *Rhs*
SYNTAX *VariableName* ::= *Rhs*
SYNTAX *MethodName* ::= `<init>`
| *Rhs*
SYNTAX *FormalParams* ::= *List{TypeName,","}*
SYNTAX *Params* ::= *List{Variable,","}*
SYNTAX *Selector* ::= *MethodName* `(` *FormalParams* `)V`
SYNTAX *TypeName* ::= *Rhs*
| *Rhs* `/` *TypeName*
SYNTAX *TypeReference* ::= `<` *Rhs* `,` *TypeName* `>`
SYNTAX *FieldReference* ::= `<` *Rhs* `,` *TypeName* `,` *Rhs* `,` *TypeReference* `>`
SYNTAX *MethodReference* ::= `<` *Rhs* `,` *TypeName* `,` *Selector* `>`
| `none`
SYNTAX *NewInstructionBase* ::= *Variable* `=new` *TypeReference* `@` *Rhs*
SYNTAX *NewInstruction* ::= *NewInstructionBase*
| *NewInstructionBase* `(` *Params* `)`
SYNTAX *GetInstruction* ::= *Variable* `=getfield` *FieldReference* *Variable*
| *Variable* `=getstatic` *FieldReference*
SYNTAX *PutInstruction* ::= `putfield` *Variable* `=` *Variable* *FieldReference*
| `putstatic` *Variable* *FieldReference*
SYNTAX *PhiInstruction* ::= *Variable* `=phi(` *Params* `)`
SYNTAX *PhiPiInstruction* ::= *Variable* `=phipi(` *Params* `)`
SYNTAX *InvokeSpecialInstruction* ::= `invokespecial` *MethodReference* *Params* `@` *Rhs* `exception:` *Variable*
SYNTAX *Instruction* ::= *NewInstruction*
| *GetInstruction*
| *PutInstruction*
| *PhiInstruction*
| *PhiPiInstruction*
| `return`
| *InvokeSpecialInstruction*
| `noinstruction`
SYNTAX *BBEdge* ::= *Rhs* `->` *Rhs* `;`
SYNTAX *BlockBody* ::= *List{Instruction,";"}*
SYNTAX *Block* ::= *Rhs* `:{` *BlockBody* `}`
SYNTAX *TaskUnit* ::= *BBEdge*
| *Block*
SYNTAX *Task* ::= *TaskUnit*
| *Task* *Task*
SYNTAX *Program* ::= *MethodDefinition*
| `start`
| `analysis`
| `done`
| *Program* `+++` *Program*
SYNTAX *MethodDefinition* ::= *MethodReference* `{` *Task* `}`

END MODULE

MODULE KWALA

IMPORTS KWALA-SYNTAX

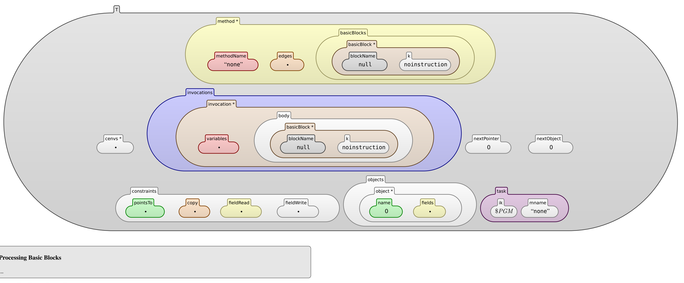
Semantics

...

Configuration

...

CONFIGURATION:



Processing Basic Blocks

...

RULE $T_1 \quad T_2 \Rightarrow T_1 \curvearrowright T_2$

RULE $I_1 ; BB_2 \Rightarrow I_1 \curvearrowright BB_2$

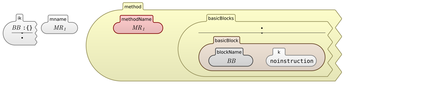
RULE $P_2 \texttt{ +++ } P_{2'} \Rightarrow P_{2'} \curvearrowright P_{2'}$

SYNTAX *Listbm* ::= `[` *Rhs* `,` *Rhs* `]`

RULE


RULE


RULE


RULE


RULE

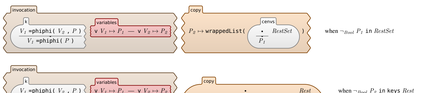

RULE


Gathering Constraints

...

SYNTAX *K* ::= `wrappedList(` *Bag* `)`

Phi functions

RULE


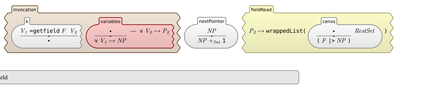
RULE


RULE


RULE


Get field

SYNTAX *FieldToPointer* ::= `(` *FieldReference* `|>` *Rhs* `)`

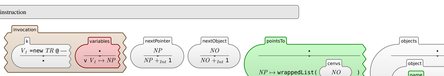
RULE


RULE


Put field

RULE


when $\neg_{Bool}\ P_1$ in keys $Rest$

RULE


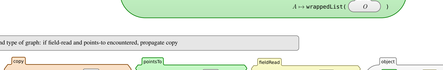
New instruction

RULE


Resolving Constraints

First type of graph: if copy encountered, propagate points-to

RULE


RULE


Second type of graph: if field-read and points-to encountered, propagate copy

RULE


RULE


Third type of graph: if field-write and points-to encountered, propagate copy

RULE


RULE

END MODULE