MODULE KWALA-SYNTAX

Syntax

...

SYNTAX *Hd* ::= `object`
| `null`
SYNTAX *Variable* ::= *Hnt*
SYNTAX *VariableName* ::= *v Hnt*
SYNTAX *Selector* ::= `<init>()V`
SYNTAX *TypeName* ::= *Hd*
| *Hd* / *TypeName*
SYNTAX *TypeReference* ::= `<` *Hd* `,` *TypeName* `>`
SYNTAX *FieldReference* ::= `<` *Hd* `,` *TypeName* `,` *Hd* `,` *TypeReference* `>`
SYNTAX *MethodReference* ::= `<` *Hd* `,` *TypeName* `,` *Selector* `>`
SYNTAX *Params* ::= *List*`[`*Variable*`,`"`,"`*`]`
SYNTAX *NewInstructionBase* ::= *Variable* `=new` *TypeReference* `@` *Hnt*
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 *PhiPhiInstruction* ::= *Variable* `=phiphi(` *Params* `)`
SYNTAX *InvokeSpecialInstruction* ::= `invokespecial` *MethodReference* *Params* `@` *Hnt* `exception:` *Variable*
SYNTAX *Instruction* ::= *NewInstruction*
| *GetInstruction*
| *PutInstruction*
| *PhiInstruction*
| *PhiPhiInstruction*
| `return`
| *InvokeSpecialInstruction*
| `noinstruction`
| `main`
SYNTAX *BBEdge* ::= *Hd* `->` *Hd* `;`
SYNTAX *BlockBody* ::= *List*`[`*Instruction*`,`"`;"`*`]`
SYNTAX *Block* ::= *Hd* `:{` *BlockBody* `}`
SYNTAX *TaskUnit* ::= *BBEdge*
| *Block*
| `start`
| `analysis`
| `done`
SYNTAX *Task* ::= *TaskUnit*
| *Task* *Task*

END MODULE

MODULE KWALA
IMPORTS KWALA-SYNTAX

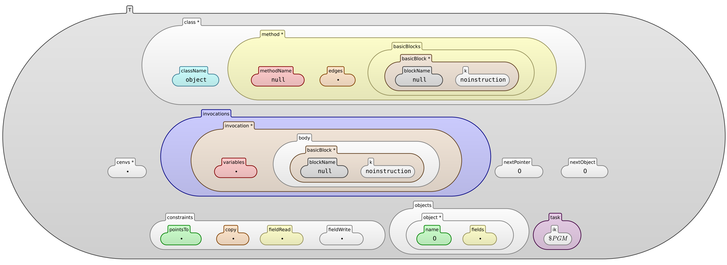
Semantics

...

Configuration

...

CONFIGURATION:



Processing Basic Blocks

...

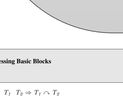
RULE $T_1 \;\; T_2 \Rightarrow T_1 \curvearrowright T_2$

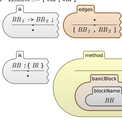
RULE $l_1 \; ; \; BB_2 \Rightarrow l_1 \curvearrowright BB_2$

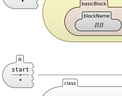
SYNTAX *ListItem* ::= `[` *Hd* `,` *Hd* `]`

RULE


RULE

RULE


RULE


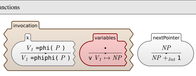
RULE


Gathering Constraints

...

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

Phi functions

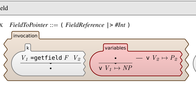
RULE


RULE


when $\neg_{Bool}\ P_1$ in *RestSet*

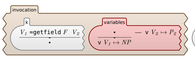
RULE


when $\neg_{Bool}\ P_2$ in keys *Rest*

RULE


Get field

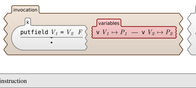
SYNTAX *FieldToPointer* ::= `{` *FieldReference* `|>` *Hnt* `}`

RULE


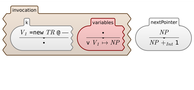
when $\neg_{Bool}\ P_2$ in keys *Rest*

RULE


Put field

RULE


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

RULE


New instruction

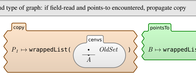
RULE


Resolving Constraints

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

RULE

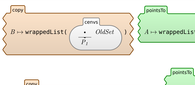

when $\neg_{Bool}\ O$ in *OldSet*

RULE


when $\neg_{Bool}\ A$ in keys *RestOfTheWorld*

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

when $\neg_{Bool}\ A$ in *OldSet*

RULE


when $\neg_{Bool}\ P_1$ in keys *OldMap*

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

when $\neg_{Bool}\ P_1$ in *OldSet*

when $\neg_{Bool}\ B$ in keys *OldMap*

END MODULE