MODULE **KWALA-SYNTAX**

Syntax

...

```
SYNTAX  #Id ::= object
              | null
              | on
              | off
SYNTAX  Variable ::= #Int
SYNTAX  VariableName ::= #Int
SYNTAX  MethodName ::= init
                     | #Id
                     | main
SYNTAX  FormalParams ::= List{TypeName,","}
SYNTAX  Params ::= List{Variable,","}
SYNTAX  Selector ::= MethodName "(" FormalParams ")"
SYNTAX  TypeName ::= #Id
                   | #Id "<" TypeName ">"
SYNTAX  TypeReference ::= #Id , TypeName >
SYNTAX  FieldReference ::= <#Id , TypeName , #Id , TypeReference >
SYNTAX  MethodReference ::= <{ #Id , TypeName , Selector }>
                          | none
SYNTAX  MethodDeclaration ::= MethodReference "(" Params ")"
SYNTAX  NewInstructionBase ::= Variable "=new TypeReference #Int
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  PhiPlusInstruction ::= Variable "=phiplus" "(" Params ")"
SYNTAX  InvokeSpecialInstruction ::= invokespecial MethodReference  Params
SYNTAX  Instruction ::= NewInstruction
                      | GetInstruction
                      | PutInstruction
                      | PhiPlusInstruction
                      | PhiPhiInstruction
                      | return
                      | InvokeSpecialInstruction
                      | if-instruction
SYNTAX  BBEdge ::= #Int
SYNTAX  BlockBody ::= List{Instruction,";"}
SYNTAX  Block ::= #Id "{" BlockBody "}"
SYNTAX  TaskUnit ::= BBEdge
                   | Block
SYNTAX  Task ::= TaskUnit
               | Task  Task
SYNTAX  Program ::= MethodDefinition
                  | start
                  | analysis
                  | done
                  | Program *** Program
SYNTAX  MethodDefinition ::= MethodDeclaration "{" Task "}"
```

END MODULE

MODULE **KWALA**
IMPORTS **KWALA-SYNTAX**

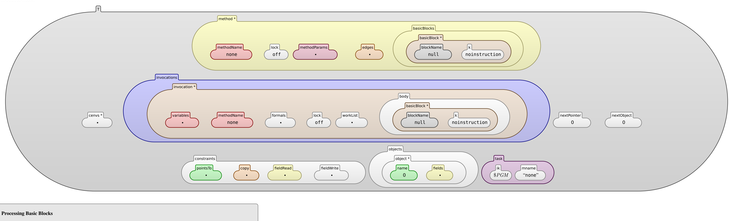
Semantics

...

Configuration

...

CONFIGURATION:



Processing Basic Blocks

...

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

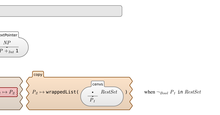
RULE $I_1 \; ; \; BB_2 \Rightarrow I_1 \curvearrowright BB_2$

RULE $P_0 \; *** \; P_0 \Rightarrow P_0 \curvearrowright P_0$

SYNTAX *Listbow* ::= { *#Id* , *#Id* }

RULE $\frac{BB_1 \; \texttt{->} \; BB_2 \; ;}{\cdot}$ mname MB_1

RULE $\frac{BB \; \{ \; IR \; \}}{\cdot}$ mname MB_1



RULE $\frac{BB \; \{\} \; ;}{\cdot}$ mname MB_1

RULE $\frac{MB_1 \; [\; PMS \;] \; \{ \; T_2 \; \}}{T_2}$ mname $\frac{\cdot}{MB_1}$

RULE $\frac{start \; ;}{\cdot}$

RULE $\frac{analysis \; ;}{\cdot}$ mname $\frac{\cdot}{none}$

Gathering Constraints

...

SYNTAX $K ::= \texttt{wrappedList}(\; Bag \;)$

Phi functions

RULE $\frac{\begin{array}{c} V_1 \texttt{=phi}(\; P \;) \\ V_1 \texttt{=phiplus}(\; P \;) \end{array}}{\cdot}$

RULE $\frac{\begin{array}{c} V_1 \texttt{=phiplus}(\; V_2 \;,\; P \;) \\ V_1 \texttt{=phiplus}(\; P \;) \end{array}}{\cdot}$

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

RULE $\frac{\begin{array}{c} V_1 \texttt{=phiplus}(\; V_2 \;,\; P \;) \\ V_1 \texttt{=phiplus}(\; P \;) \end{array}}{\cdot}$

when $\neg_{Bool}$ P_0 in keys *Bool*

RULE $\frac{V_1 \texttt{=phi}(\;)}{\cdot}$

Get field

SYNTAX *FieldToPointer* ::= { *FieldReference* > *#Int* }

RULE $\frac{\begin{array}{c} V_1 \texttt{=getfield}\; F \;\; V_2 \\ \cdot \\ V_1 \mapsto NP \end{array}}{\cdot}$

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

RULE $\frac{\begin{array}{c} V_1 \texttt{=getfield}\; F \;\; V_2 \\ \cdot \\ V_1 \mapsto NP \end{array}}{\cdot}$

Put field

RULE $\frac{\begin{array}{c} putfield\; V_1 = V_2 \;\; F \\ \cdot \\ \cdot \end{array}}{\cdot}$

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

RULE $\frac{\begin{array}{c} putfield\; V_1 = V_2 \;\; F \\ \cdot \\ \cdot \end{array}}{\cdot}$

New instruction

RULE $\frac{\begin{array}{c} V_1 \texttt{=new}\; TR\; @ \; \cdot \\ \cdot \\ V_1 \mapsto NP \end{array}}{\cdot}$

Invocation

Never seen before: look for it in <methods>

RULE $\frac{invokespecial\; MR_1 \;\; PMS}{\cdot}$

RULE

RULE

Seen before: look for it in <invocations>

RULE

RULE

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

RULE

when $\neg_{Bool}$ P_0 in keys *Bool*

RULE $\frac{invokespecial \; \cdot \; \cdot}{\cdot}$

Resolving Constraints

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

RULE $\frac{B \mapsto \texttt{wrappedList}(\; A \;)}{\cdot}$

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

RULE $\frac{B \mapsto \texttt{wrappedList}(\; A \;)}{\cdot}$

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

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

RULE $\frac{\begin{array}{c} P_1 \mapsto \texttt{wrappedList}(\; A \;) \\ B \mapsto \texttt{wrappedList}(\; O \;) \\ B \mapsto \texttt{wrappedList}(\; \{ \; F \; [\texttt{=}\; A \;] \; \} \end{array}}{\cdot}$

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

RULE

when $\neg_{Bool}$ P_0 in keys *OldMap*

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

RULE $\frac{\begin{array}{c} B \mapsto \texttt{wrappedList}(\; P_1 \;) \\ A \mapsto \texttt{wrappedList}(\; O \;) \\ A \mapsto \texttt{wrappedList}(\; \{ \; F \; [\texttt{=}\; B \;] \; \} \end{array}}{\cdot}$

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

RULE

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

END MODULE