Unspecified Veriopt Theory

January 8, 2022

Contents

theory *TreeSnippets*
imports
Semantics.TreeToGraphThms
Veriopt.Snipping
begin

notation (*latex*)
kind (-⟪-⟫)

notation (*latex*)
IRTreeEval.ord-IRExpr-inst.less-eq-IRExpr (- $\longmapsto$ -)

abstract-syntax-tree

datatype *IRExpr* = *UnaryExpr IRUnaryOp IRExpr*
| *BinaryExpr IRBinaryOp IRExpr IRExpr*
| *ConditionalExpr IRExpr IRExpr IRExpr*
| *ParameterExpr nat Stamp*
| *LeafExpr nat Stamp* | *ConstantExpr Value*
| *ConstantVar* (*char list*)
| *VariableExpr* (*char list*) *Stamp*

tree-semantics

semantics:constant semantics:parameter semantics:conditional semantics:unary semantics:binary semantics:leaf

tree-evaluation-deterministic

$[\![[m,p] \vdash e \mapsto v1;\ [m,p] \vdash e \mapsto v2]\!] \Longrightarrow v1 = v2$

expression-refinement

$$(e2 \leq e1) = (\forall\, m\ p\ v.\ [m,p] \vdash e1 \mapsto v \longrightarrow [m,p] \vdash e2 \mapsto v)$$

expression-refinement-monotone

$$e' \leq e \Longrightarrow \mathit{UnaryExpr}\ op\ e' \leq \mathit{UnaryExpr}\ op\ e$$
$$[\![x' \leq x;\ y' \leq y]\!] \Longrightarrow \mathit{BinaryExpr}\ op\ x'\ y' \leq \mathit{BinaryExpr}\ op\ x\ y$$
$$[\![ce' \leq ce;\ te' \leq te;\ fe' \leq fe]\!] \Longrightarrow \mathit{ConditionalExpr}\ ce'\ te'\ fe' \leq \mathit{ConditionalExpr}\ ce\ te\ fe$$

graph-representation

typedef $\mathit{IRGraph} = \{g :: \mathit{ID} \rightharpoonup \mathit{IRNode}\ .\ \mathit{finite}\ (\mathit{dom}\ g)\}$

graph2tree

semantics:constant semantics:parameter semantics:conditional semantics:unary semantics:convert semantics:binary semantics:leaf *preeval*

is-preevaluated (*InvokeNode n uu uv uw ux uy*) = *True*
is-preevaluated (*InvokeWithExceptionNode n uz va vb vc vd ve*) = *True*
is-preevaluated (*NewInstanceNode n vf vg vh*) = *True*
is-preevaluated (*LoadFieldNode n vi vj vk*) = *True*
is-preevaluated (*SignedDivNode n vl vm vn vo vp*) = *True*
is-preevaluated (*SignedRemNode n vq vr vs vt vu*) = *True*
is-preevaluated (*ValuePhiNode n vv vw*) = *True*
is-preevaluated (*AbsNode v*) = *False*
is-preevaluated (*AddNode v va*) = *False*
is-preevaluated (*AndNode v va*) = *False*
is-preevaluated (*BeginNode v*) = *False*
is-preevaluated (*BytecodeExceptionNode v va vb*) = *False*
is-preevaluated (*ConditionalNode v va vb*) = *False*
is-preevaluated (*ConstantNode v*) = *False*
is-preevaluated (*DynamicNewArrayNode v va vb vc vd*) = *False*
is-preevaluated *EndNode* = *False*
is-preevaluated (*ExceptionObjectNode v va*) = *False*
is-preevaluated (*FrameState v va vb vc*) = *False*
is-preevaluated (*IfNode v va vb*) = *False*
is-preevaluated (*IntegerBelowNode v va*) = *False*
is-preevaluated (*IntegerEqualsNode v va*) = *False*
is-preevaluated (*IntegerLessThanNode v va*) = *False*
is-preevaluated (*IsNullNode v*) = *False*
is-preevaluated (*KillingBeginNode v*) = *False*
is-preevaluated (*LeftShiftNode v va*) = *False*
is-preevaluated (*LogicNegationNode v*) = *False*
is-preevaluated (*LoopBeginNode v va vb vc*) = *False*
is-preevaluated (*LoopEndNode v*) = *False*
is-preevaluated (*LoopExitNode v va vb*) = *False*
is-preevaluated (*MergeNode v va vb*) = *False*
is-preevaluated (*MethodCallTargetNode v va*) = *False*
is-preevaluated (*MulNode v va*) = *False*
is-preevaluated (*NarrowNode v va vb*) = *False*
is-preevaluated (*NegateNode v*) = *False*
is-preevaluated (*NewArrayNode v va vb*) = *False*
is-preevaluated (*NotNode v*) = *False*
is-preevaluated (*OrNode v va*) = *False*

is-preevaluated (*ParameterNode v*) = *False*
is-preevaluated (*PiNode v va*) = *False*
is-preevaluated (*ReturnNode v va*) = *False*
is-preevaluated (*RightShiftNode v va*) = *False*
is-preevaluated (*ShortCircuitOrNode v va*) = *False*
is-preevaluated (*SignExtendNode v va vb*) = *False*

deterministic-representation

$$[\![g \vdash n \simeq e1;\ g \vdash n \simeq e2]\!] \Longrightarrow e1 = e2$$

graph-semantics

$$([g,m,p] \vdash n \mapsto v) = (\exists e.\ g \vdash n \simeq e \wedge [m,p] \vdash e \mapsto v)$$

graph-semantics-deterministic

$$[g,m,p] \vdash nid \mapsto v1 \wedge [g,m,p] \vdash nid \mapsto v2 \Longrightarrow v1 = v2$$

graph-refinement

graph-refinement g1 g2 =
$(\forall n.\ n \in ids\ g1 \longrightarrow$
$\quad (\forall e1.\ g1 \vdash n \simeq e1 \longrightarrow (\exists e2.\ g2 \vdash n \simeq e2 \wedge e2 \leq e1)))$

translations
n <= *CONST as-set n*

experiment begin

Experimental embedding into a simplier but usable form for expression nodes in a graph

datatype *ExprIRNode* =
ExprUnaryNode IRUnaryOp ID |
ExprBinaryNode IRBinaryOp ID ID |
ExprConditionalNode ID ID ID |
ExprConstantNode Value |
ExprParameterNode nat |
ExprLeafNode ID |
NotExpr

fun *embed-expr* :: *IRNode* ⇒ *ExprIRNode* **where**
embed-expr (*ConstantNode v*) = *ExprConstantNode v* |
embed-expr (*ParameterNode i*) = *ExprParameterNode i* |
embed-expr (*ConditionalNode c t f*) = *ExprConditionalNode c t f* |
embed-expr (*AbsNode x*) = *ExprUnaryNode UnaryAbs x* |
embed-expr (*NotNode x*) = *ExprUnaryNode UnaryNot x* |

embed-expr (*NegateNode x*) = *ExprUnaryNode UnaryNeg x* |
embed-expr (*LogicNegationNode x*) = *ExprUnaryNode UnaryLogicNegation x* |
embed-expr (*AddNode x y*) = *ExprBinaryNode BinAdd x y* |
embed-expr (*MulNode x y*) = *ExprBinaryNode BinMul x y* |
embed-expr (*SubNode x y*) = *ExprBinaryNode BinSub x y* |
embed-expr (*AndNode x y*) = *ExprBinaryNode BinAnd x y* |
embed-expr (*OrNode x y*) = *ExprBinaryNode BinOr x y* |
embed-expr (*XorNode x y*) = *ExprBinaryNode BinXor x y* |
embed-expr (*IntegerBelowNode x y*) = *ExprBinaryNode BinIntegerBelow x y* |
embed-expr (*IntegerEqualsNode x y*) = *ExprBinaryNode BinIntegerEquals x y* |
embed-expr (*IntegerLessThanNode x y*) = *ExprBinaryNode BinIntegerLessThan x y* |
embed-expr (*NarrowNode ib rb x*) = *ExprUnaryNode* (*UnaryNarrow ib rb*) *x* |
embed-expr (*SignExtendNode ib rb x*) = *ExprUnaryNode* (*UnarySignExtend ib rb*) *x* |
embed-expr (*ZeroExtendNode ib rb x*) = *ExprUnaryNode* (*UnaryZeroExtend ib rb*) *x* |
embed-expr - = *NotExpr*

lemma *unary-embed*:
assumes $g \vdash n \simeq$ *UnaryExpr op x*
shows $\exists\ x'$. *embed-expr* (*kind g n*) = *ExprUnaryNode op* x'
using *assms* **by** (*induction UnaryExpr op x rule*: *rep.induct*; *simp*)

lemma *equal-embedded-x*:
assumes $g \vdash n \simeq$ *UnaryExpr op xe*
assumes *embed-expr* (*kind g n*) = *ExprUnaryNode* op' *x*
shows $g \vdash x \simeq xe$
using *assms* **by** (*induction UnaryExpr op xe rule*: *rep.induct*; *simp*)

lemma *blah*:
assumes *embed-expr* (*kind g n*) = *ExprUnaryNode op* n'
assumes $g \vdash n' \simeq e$
shows ($g \vdash n \simeq$ *UnaryExpr op e*)
using *assms*(*2*,*1*) **apply** (*cases kind g n*; *auto*)
using *rep.AbsNode* **apply** *blast*
using *rep.LogicNegationNode* **apply** *blast*
using *NarrowNode* **apply** *presburger*
using *rep.NegateNode* **apply** *blast*
using *rep.NotNode* **apply** *blast*
using *rep.SignExtendNode* **apply** *blast*
using *rep.ZeroExtendNode* **by** *blast*
end

graph-semantics-preservation

$[\![e2' \leq e1'; \{n'\} \trianglelefteq g1 \subseteq g2;$
$g1 \vdash n' \simeq e1'; g2 \vdash n' \simeq e2']\!]$
$\Longrightarrow \textit{graph-refinement } g1\ g2$

maximal-sharing

$\textit{maximal-sharing } g =$
$(\forall n1\ n2.$
$\quad n1 \in \textit{ids } g \wedge n2 \in \textit{ids } g \longrightarrow$
$\quad (\forall e.\ g \vdash n1 \simeq e \wedge g \vdash n2 \simeq e \longrightarrow n1 = n2))$

tree-to-graph-rewriting

$e2 \leq e1 \wedge$
$g1 \vdash n \simeq e1 \wedge$
$\textit{maximal-sharing } g1 \wedge$
$\{n\} \trianglelefteq g1 \subseteq g2 \wedge$
$g2 \vdash n \simeq e2 \wedge \textit{maximal-sharing } g2 \Longrightarrow$
$\textit{graph-refinement } g1\ g2$

graph-represents-expression

$(g \vdash n \trianglelefteq e) = (\forall m\ p\ v.\ [m,p] \vdash e \mapsto v \longrightarrow [g,m,p] \vdash n \mapsto v)$

graph-construction

$e2 \leq e1 \wedge$
$g1 \subseteq g2 \wedge$
$\textit{maximal-sharing } g1 \wedge$
$g2 \vdash n \simeq e2 \wedge \textit{maximal-sharing } g2 \Longrightarrow$
$g2 \vdash n \trianglelefteq e1 \wedge \textit{graph-refinement } g1\ g2$

end
theory *SlideSnippets*
imports
Semantics.TreeToGraphThms
Veriopt.Snipping
begin

notation (*latex*)
  kind (-⟪-⟫)

notation (*latex*)
  IRTreeEval.ord-IRExpr-inst.less-eq-IRExpr (- ⟼ -)

abstract-syntax-tree

datatype *IRExpr* =
  UnaryExpr IRUnaryOp IRExpr
  | *BinaryExpr IRBinaryOp IRExpr IRExpr*
  | *ConditionalExpr IRExpr IRExpr IRExpr*
  | *ParameterExpr nat Stamp*
  | *LeafExpr nat Stamp*
  | *ConstantExpr Value*
  | *ConstantVar* (*char list*)
  | *VariableExpr* (*char list*) *Stamp*

tree-semantics

semantics:constant semantics:parameter semantics:unary semantics:binary semantics:leaf

expression-refinement

$$(e2 \leq e1) = (\forall m\ p\ v.\ [m,p] \vdash e1 \mapsto v \longrightarrow [m,p] \vdash e2 \mapsto v)$$

graph2tree

semantics:constant semantics:unary semantics:binary

graph-semantics

$$([g,m,p] \vdash n \mapsto v) = (\exists e.\ g \vdash n \simeq e \wedge [m,p] \vdash e \mapsto v)$$

graph-refinement

graph-refinement g1 g2 =
($\forall n.\ n \in ids\ g1 \longrightarrow$
    $(\forall e1.\ g1 \vdash n \simeq e1 \longrightarrow (\exists e2.\ g2 \vdash n \simeq e2 \wedge e2 \leq e1)))$

translations
n <= $CONST$ $as\text{-}set$ n

graph-semantics-preservation

$$[\![e2' \leq e1'; \{n'\} \trianglelefteq g1 \subseteq g2; \; g1 \vdash n' \simeq e1'; g2 \vdash n' \simeq e2']\!] \Longrightarrow graph\text{-}refinement\ g1\ g2$$

maximal-sharing

$$maximal\text{-}sharing\ g = (\forall\, n1\ n2.\ n1 \in ids\ g \wedge n2 \in ids\ g \longrightarrow (\forall\, e.\ g \vdash n1 \simeq e \wedge g \vdash n2 \simeq e \longrightarrow n1 = n2))$$

tree-to-graph-rewriting

$$e2 \leq e1 \wedge g1 \vdash n \simeq e1 \wedge maximal\text{-}sharing\ g1 \wedge \{n\} \trianglelefteq g1 \subseteq g2 \wedge g2 \vdash n \simeq e2 \wedge maximal\text{-}sharing\ g2 \Longrightarrow graph\text{-}refinement\ g1\ g2$$

graph-represents-expression

$$(g \vdash n \trianglelefteq e) = (\forall\, m\ p\ v.\ [m,p] \vdash e \mapsto v \longrightarrow [g,m,p] \vdash n \mapsto v)$$

graph-construction

$$e2 \leq e1 \wedge g1 \subseteq g2 \wedge maximal\text{-}sharing\ g1 \wedge g2 \vdash n \simeq e2 \wedge maximal\text{-}sharing\ g2 \Longrightarrow g2 \vdash n \trianglelefteq e1 \wedge graph\text{-}refinement\ g1\ g2$$

end