Veriopt Theories

March 18, 2022

Contents

theory *TreeSnippets*
  imports
    Canonicalizations.ConditionalPhase
    Optimizations.CanonicalizationSyntax
    Semantics.TreeToGraphThms
    Snippets.Snipping
    HOL−Library.OptionalSugar
begin

no-notation *ConditionalExpr* (- *?* - : -)

notation (*latex*)
  kind (-《-》)

notation (*latex*)
  valid-value (- ∈ -)

notation (*latex*)
  val-to-bool (*bool-of* -)

notation (*latex*)
  constantAsStamp (*stamp-from-value* -)

notation (*latex*)
  size (*trm(-)*)

translations
  $y > x <= x < y$

notation (*latex*)
  greater (- >/ -)

translations

$n <= CONST\ Rep\text{-}intexp\ n$
$n <= CONST\ Rep\text{-}i32exp\ n$
$n <= CONST\ Rep\text{-}i64exp\ n$

lemma *vminusv*: $\forall\, vv\ v\ .\ vv = IntVal32\ v \longrightarrow v - v = 0$
by *simp*
thm-oracles *vminusv*

lemma *vminusv2*: $\forall\ v{::}int32\ .\ v - v = 0$
by *simp*

lemma *redundant-sub*:
$\forall\, vv_1\ vv_2\ v_1\ v_2\ .\ vv_1 = IntVal32\ v_1 \wedge vv_2 = IntVal32\ v_2 \longrightarrow v_1 - (v_1 - v_2) = v_2$
by *simp*
thm-oracles *redundant-sub*

lemma *redundant-sub2*:
$\forall\ (v_1{::}int32)\ (v_2{::}int32)\ .\ v_1 - (v_1 - v_2) = v_2$
by *simp*

val-eq

$\forall\, vv\ v.\ vv = IntVal32\ v \longrightarrow v - v = 0$

$\forall\, vv_1\ vv_2\ v_1\ v_2.\ vv_1 = IntVal32\ v_1 \wedge vv_2 = IntVal32\ v_2 \longrightarrow v_1 - (v_1 - v_2) = v_2$

phase *tmp*
terminating *size*
begin

sub-same-32

optimization *sub-same-32*: $(e{::}i32exp) - e \longmapsto const\ (IntVal32\ 0)$

apply (*unfold rewrite-preservation.simps*, *unfold rewrite-termination.simps*, *rule conjE*, *simp*) **apply** *auto*[*1*] **using** *Rep-i32exp evalDet is-IntVal32-def*
apply (*smt* (*verit*, *del-insts*) *eq-iff-diff-eq-0 evaltree.simps int-constants-valid int-val-sub.simps*(*1*) *is-int-val.simps*(*1*) *mem-Collect-eq*)
unfolding *size.simps*
by (*metis add-strict-increasing gr-implies-not0 less-one linorder-not-le size-gt-0*)

sub-same-64

optimization *sub-same-64*: $(e{::}i64exp) - e \longmapsto const\ (IntVal64\ 0)$

apply *auto*
apply (*metis* (*no-types*, *opaque-lifting*) *ConstantExpr bin-eval.simps*(*3*) *bin-eval-preserves-validity cancel-comm-monoid-add-class.diff-cancel evalDet i64e-eval int-and-equal-bits.simps*(*2*)

intval-sub.simps(*2*))
  by (*simp add*: *Suc-le-eq add-strict-increasing size-gt-0*)
end

thm-oracles *sub-same-32*

ast-example

BinaryExpr BinAdd (*BinaryExpr BinMul x x*) (*BinaryExpr BinMul x x*)

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*

value

datatype *Value* = *UndefVal*
  | *IntVal32* (*32 word*)
  | *IntVal64* (*64 word*)
  | *ObjRef* (*nat option*)
  | *ObjStr* (*char list*)

eval

unary-eval :: *IRUnaryOp* ⇒ *Value* ⇒ *Value*

bin-eval :: *IRBinaryOp* ⇒ *Value* ⇒ *Value* ⇒ *Value*

tree-semantics

semantics:unary semantics:binary semantics:conditional semantics:constant semantics:parameter semantics:leaf *tree-evaluation-deterministic*

$[m,p] \vdash e \mapsto v_1 \wedge [m,p] \vdash e \mapsto v_2 \Longrightarrow v_1 = v_2$

thm-oracles *evalDet*

expression-refinement

$e_1 \sqsupseteq e_2 = (\forall\, m\ p\ v.\ [m,p] \vdash e_1 \mapsto v \longrightarrow [m,p] \vdash e_2 \mapsto v)$

expression-refinement-monotone

$$\begin{array}{lll} e \sqsupseteq e' & \Longrightarrow & \mathit{UnaryExpr}\ op\ e \sqsupseteq \mathit{UnaryExpr}\ op\ e' \\ x \sqsupseteq x' \wedge y \sqsupseteq y' & \Longrightarrow & \mathit{BinaryExpr}\ op\ x\ y \sqsupseteq \mathit{BinaryExpr}\ op\ x'\ y' \\ ce \sqsupseteq ce' \wedge te \sqsupseteq te' \wedge fe \sqsupseteq fe' & \Longrightarrow & \mathit{ConditionalExpr}\ ce\ te\ fe \sqsupseteq \mathit{ConditionalExpr}\ ce'\ te'\ fe' \end{array}$$

ML ‹
```
(*fun get-list (phase: phase option) =
  case phase of
    NONE => [] |
    SOME p => (#rewrites p)

fun get-rewrite name thy =
  let
    val (phases, lookup) = (case RWList.get thy of
      NoPhase store => store |
      InPhase (name, store, -) => store)
    val rewrites = (map (fn x => get-list (lookup x)) phases)
  in
    rewrites
  end

fun rule-print name =
  Document-Output.antiquotation-pretty name (Args.term)
    (fn ctxt => fn (rule) => (*Pretty.str hello*)
      Pretty.block (print-all-phases (Proof-Context.theory-of ctxt)));
(*
      Goal-Display.pretty-goal
        (Config.put Goal-Display.show-main-goal main ctxt)
        (#goal (Proof.goal (Toplevel.proof-of (Toplevel.presentation-state ctxt)))));
*)

val - = Theory.setup
 (rule-print binding‹rule›);*)
```
›

phase *SnipPhase*
  terminating *size*
begin

BinaryFoldConstant

optimization *BinaryFoldConstant*: *BinaryExpr op* (*const v1*) (*const v2*) $\longmapsto$ *ConstantExpr* (*bin-eval op v1 v2*) *when int-and-equal-bits v1 v2*

unfolding *rewrite-preservation.simps rewrite-termination.simps*
apply (*rule conjE*, *simp*, *simp del*: *le-expr-def*)

BinaryFoldConstantObligation

1. *int-and-equal-bits v1 v2* $\longrightarrow$
   BinaryExpr op (*ConstantExpr v1*) (*ConstantExpr v2*) $\sqsupseteq$
   ConstantExpr (*bin-eval op v1 v2*)
2. *int-and-equal-bits v1 v2* $\longrightarrow$
   trm(BinaryExpr op (*ConstantExpr v1*) (*ConstantExpr v2*)*)* >
   trm(ConstantExpr (*bin-eval op v1 v2*)*)*

using *BinaryFoldConstant* **by** *auto*

AddCommuteConstantRight

optimization *AddCommuteConstantRight*: ((*const v*) + *y*) $\longmapsto$ *y* + (*const v*) *when* ¬(*is-ConstantExpr y*)

unfolding *rewrite-preservation.simps rewrite-termination.simps*
apply (*rule conjE*, *simp*, *simp del*: *le-expr-def*)

AddCommuteConstantRightObligation

1. ¬ *is-ConstantExpr y* $\longrightarrow$
   BinaryExpr BinAdd (*ConstantExpr v*) *y* $\sqsupseteq$
   BinaryExpr BinAdd y (*ConstantExpr v*)
2. ¬ *is-ConstantExpr y* $\longrightarrow$
   trm(BinaryExpr BinAdd (*ConstantExpr v*) *y)* >
   trm(BinaryExpr BinAdd y (*ConstantExpr v*)*)*

using *AddShiftConstantRight* **by** *auto*

AddNeutral

optimization *AddNeutral*: ((*e*::*i32exp*) + (*const* (*IntVal32 0*))) $\longmapsto$ *e*

unfolding *rewrite-preservation.simps rewrite-termination.simps*

apply (*rule conjE*, *simp*, *simp del*: *le-expr-def*)

AddNeutralObligation

1. *BinaryExpr BinAdd e* (*ConstantExpr* (*IntVal32 0*)) $\sqsupseteq$ *e*
2. *trm(BinaryExpr BinAdd e* (*ConstantExpr* (*IntVal32 0*))*)* > *trm(e)*

using *neutral-zero*(*1*) *rewrite-preservation.simps*(*1*) **apply** *blast*
by *auto*

InverseLeftSub

optimization *InverseLeftSub*: $((e_1::intexp) - (e_2::intexp)) + e_2 \longmapsto e_1$

unfolding *rewrite-preservation.simps rewrite-termination.simps*
apply (*rule conjE*, *simp*, *simp del*: *le-expr-def*)

InverseLeftSubObligation

1. *BinaryExpr BinAdd* (*BinaryExpr BinSub* e_1 e_2) $e_2 \sqsupseteq e_1$
2. *trm(BinaryExpr BinAdd* (*BinaryExpr BinSub* e_1 e_2) e_2*)* > *trm*(e_1)

using *neutral-left-add-sub* **by** *auto*

InverseRightSub

optimization *InverseRightSub*: $(e_2::intexp) + ((e_1::intexp) - e_2) \longmapsto e_1$

unfolding *rewrite-preservation.simps rewrite-termination.simps*
apply (*rule conjE*, *simp*, *simp del*: *le-expr-def*)

InverseRightSubObligation

1. *BinaryExpr BinAdd* e_2 (*BinaryExpr BinSub* e_1 e_2) $\sqsupseteq e_1$
2. *trm(BinaryExpr BinAdd* e_2 (*BinaryExpr BinSub* e_1 e_2)*)* > *trm*(e_1)

using *neutral-right-add-sub* **by** *auto*

AddToSub

optimization *AddToSub*: $-e + y \longmapsto y - e$

unfolding *rewrite-preservation.simps rewrite-termination.simps*
apply (*rule conjE*, *simp*, *simp del*: *le-expr-def*)

AddToSubObligation

1. *BinaryExpr BinAdd* (*UnaryExpr UnaryNeg e*) *y* $\sqsupseteq$ *BinaryExpr BinSub y e*
2. *trm(BinaryExpr BinAdd* (*UnaryExpr UnaryNeg e*) *y)* > *trm(BinaryExpr BinSub y e)*

using *AddLeftNegateToSub* **by** *auto*

end

definition *trm* **where** *trm* = *size*

phase

phase *AddCanonicalizations*
terminating *trm*
begin...**end**

hide-const (**open**) *Form.wf-stamp*

phase-example

phase *Conditional*
terminating *trm*
begin

phase-example-1

optimization *negate-condition*: $(\neg e \ ?\ x : y) \longmapsto (e \ ?\ y : x)$

using *ConditionalPhase.negate-condition*
by (*auto simp*: *trm-def*)

phase-example-2

optimization *const-true*: $(true \ ?\ x : y) \longmapsto x$

by (*auto simp*: *trm-def*)

phase-example-3

optimization *const-false*: $(false \ ?\ x : y) \longmapsto y$

by (*auto simp*: *trm-def*)

phase-example-4

optimization *equal-branches*: $(e \ ?\ x : x) \longmapsto x$

by (*auto simp*: *trm-def*)

phase-example-5

optimization *condition-bounds-x*: $((u < v)\ ?\ x : y) \longmapsto x$
when (*stamp-under* (*stamp-expr u*) (*stamp-expr v*) $\wedge$ *wf-stamp u* $\wedge$ *wf-stamp v*)

using *ConditionalPhase.condition-bounds-x*(*1*)
by (*blast*, *auto simp*: *trm-def*)

phase-example-6

optimization *condition-bounds-y*: $((x < y)\ ?\ x : y) \longmapsto y$
when (*stamp-under* (*stamp-expr y*) (*stamp-expr x*) $\wedge$ *wf-stamp x* $\wedge$ *wf-stamp y*)

using *ConditionalPhase.condition-bounds-y*(*1*)
by (*blast*, *auto simp*: *trm-def*)

phase-example-7

end

termination

$$
\begin{aligned}
&trm(UnaryExpr\ op\ e) &&= trm(e) + 1\\
&trm(BinaryExpr\ BinAdd\ x\ y) &&= trm(x) + 2 * trm(y)\\
&trm(ConditionalExpr\ cond\ t\ f) &&= trm(cond) + trm(t) + trm(f) + 2\\
&trm(ConstantExpr\ c) &&= 1\\
&trm(ParameterExpr\ ind\ s) &&= 2\\
&trm(LeafExpr\ nid\ s) &&= 2
\end{aligned}
$$

graph-representation

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

graph2tree

rep:constant rep:parameter rep:conditional rep:unary rep:convert rep:binary rep:leaf rep:ref *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

$g \vdash n \simeq e_1 \wedge g \vdash n \simeq e_2 \Longrightarrow e_1 = e_2$

thm-oracles *repDet*

well-formed-term-graph

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

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

$[g,m,p] \vdash n \mapsto v_1 \wedge [g,m,p] \vdash n \mapsto v_2 \Longrightarrow v_1 = v_2$

thm-oracles *graphDet*

notation (*latex*)
graph-refinement (*term-graph-refinement* -)

graph-refinement

$\textit{term-graph-refinement}\ g_1\ g_2 =$
$(\textit{ids}\ g_1 \subseteq \textit{ids}\ g_2 \wedge$
$(\forall\, n.\ n \in \textit{ids}\ g_1 \longrightarrow (\forall\, e.\ g_1 \vdash n \simeq e \longrightarrow g_2 \vdash n \trianglelefteq e)))$

translations
n <= *CONST as-set* n

graph-semantics-preservation

${e_1}' \sqsupseteq {e_2}' \wedge$
$\{n\} \lhd g_1 \subseteq g_2 \wedge$
$g_1 \vdash n \simeq {e_1}' \wedge g_2 \vdash n \simeq {e_2}' \Longrightarrow$
$\textit{term-graph-refinement}\ g_1\ g_2$

thm-oracles *graph-semantics-preservation-subscript* *maximal-sharing*

$maximal\text{-}sharing\ g =$
$(\forall\, n_1\ n_2.$
$\quad n_1 \in true\text{-}ids\ g \wedge n_2 \in true\text{-}ids\ g \longrightarrow$
$\quad (\forall\, e.\ g \vdash n_1 \simeq e \wedge$
$\qquad g \vdash n_2 \simeq e \wedge stamp\ g\ n_1 = stamp\ g\ n_2 \longrightarrow$
$\qquad n_1 = n_2))$

tree-to-graph-rewriting

$e_1 \sqsupseteq e_2 \wedge$
$g_1 \vdash n \simeq e_1 \wedge$
$maximal\text{-}sharing\ g_1 \wedge$
$\{n\} \lhd g_1 \subseteq g_2 \wedge$
$g_2 \vdash n \simeq e_2 \wedge$
$maximal\text{-}sharing\ g_2 \Longrightarrow$
$term\text{-}graph\text{-}refinement\ g_1\ g_2$

thm-oracles *tree-to-graph-rewriting*

term-graph-refines-term

$(g \vdash n \trianglelefteq e) = (\exists\, e'.\ g \vdash n \simeq e' \wedge e \sqsupseteq e')$

term-graph-evaluation

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

graph-construction

$e_1 \sqsupseteq e_2 \wedge g_1 \subseteq g_2 \wedge g_2 \vdash n \simeq e_2 \Longrightarrow$
$g_2 \vdash n \trianglelefteq e_1 \wedge term\text{-}graph\text{-}refinement\ g_1\ g_2$

thm-oracles *graph-construction*

term-graph-reconstruction

$g \oplus e \leadsto (g', n) \Longrightarrow g' \vdash n \simeq e \wedge g \subseteq g'$

refined-insert

$e_1 \sqsupseteq e_2 \wedge g_1 \oplus e_2 \rightsquigarrow (g_2,\ n') \Longrightarrow$
$g_2 \vdash n' \trianglelefteq e_1 \wedge \textit{term-graph-refinement}\ g_1\ g_2$

end
theory *SlideSnippets*
imports
Semantics.TreeToGraphThms
Snippets.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:unary semantics:binary semantics:leaf

expression-refinement

$$e_1 \sqsupseteq e_2 = (\forall\, m\ p\ v.\ [m,p] \vdash e_1 \mapsto v \longrightarrow [m,p] \vdash e_2 \mapsto v)$$

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

$graph\text{-}refinement\ g_1\ g_2 =$
$(ids\ g_1 \subseteq ids\ g_2 \wedge$
$\ (\forall\, n.\ n \in ids\ g_1 \longrightarrow (\forall\, e.\ g_1 \vdash n \simeq e \longrightarrow g_2 \vdash n \trianglelefteq e)))$

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

graph-semantics-preservation

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

maximal-sharing

$maximal\text{-}sharing\ g =$
$(\forall\, n_1\ n_2.$
$\quad n_1 \in true\text{-}ids\ g \wedge n_2 \in true\text{-}ids\ g \longrightarrow$
$\quad (\forall\, e.\ g \vdash n_1 \simeq e \wedge$
$\qquad g \vdash n_2 \simeq e \wedge stamp\ g\ n_1 = stamp\ g\ n_2 \longrightarrow$
$\qquad n_1 = n_2))$

tree-to-graph-rewriting

$e_1 \sqsupseteq e_2 \wedge$
$g_1 \vdash n \simeq e_1 \wedge$
$maximal\text{-}sharing\ g_1 \wedge$
$\{n\} \lhd g_1 \subseteq g_2 \wedge$
$g_2 \vdash n \simeq e_2 \wedge maximal\text{-}sharing\ g_2 \Longrightarrow$
$graph\text{-}refinement\ g_1\ g_2$

graph-represents-expression

$(g \vdash n \trianglelefteq e) = (\exists\, e'.\ g \vdash n \simeq e' \land e \sqsupseteq e')$

graph-construction

$e_1 \sqsupseteq e_2 \land g_1 \subseteq g_2 \land g_2 \vdash n \simeq e_2 \Longrightarrow$
$g_2 \vdash n \trianglelefteq e_1 \land \textit{graph-refinement}\ g_1\ g_2$

end