Veriopt Theories

March 18, 2022

Contents

1 Canonicalization Phase

theory *Common*
imports
OptimizationDSL.Canonicalization
HOL−Eisbach.Eisbach
begin

fun *size* :: *IRExpr* ⇒ *nat* **where**
size (*UnaryExpr op e*) = (*size e*) + *1* |
size (*BinaryExpr BinAdd x y*) = (*size x*) + ((*size y*) ∗ *2*) |
size (*BinaryExpr op x y*) = (*size x*) + (*size y*) |
size (*ConditionalExpr cond t f*) = (*size cond*) + (*size t*) + (*size f*) + *2* |
size (*ConstantExpr c*) = *1* |
size (*ParameterExpr ind s*) = *2* |
size (*LeafExpr nid s*) = *2* |
size (*ConstantVar c*) = *2* |
size (*VariableExpr x s*) = *2*

lemma *size-pos*[*simp*]: *0* < *size y*
apply (*induction y*; *auto?*)
subgoal premises *prems* **for** *op a b*
using *prems* **by** (*induction op*; *auto*)
done

lemma *size-non-add*: *op* ≠ *BinAdd* ⟹ *size* (*BinaryExpr op a b*) = *size a* + *size b*
by (*induction op*; *auto*)

lemma *size-non-const*:
¬ *is-ConstantExpr y* ⟹ *1* < *size y*
using *size-pos* **apply** (*induction y*; *auto*)

```
  subgoal premises prems for op a b
    apply (cases op = BinAdd)
    using size-non-add size-pos apply auto
    by (simp add: Suc-lessI one-is-add)+
  done

method unfold-optimization =
  (unfold rewrite-preservation.simps, unfold rewrite-termination.simps,
    unfold intval.simps,
    rule conjE, simp, simp del: le-expr-def)
  | (unfold rewrite-preservation.simps, unfold rewrite-termination.simps,
    rule conjE, simp, simp del: le-expr-def)

end
```

1.1 Conditional Expression

```
theory ConditionalPhase
  imports
    Common
    Proofs.StampEvalThms
begin

phase Conditional
  terminating size
begin

lemma negates: is-IntVal32 e ∨ is-IntVal64 e ⟹ val-to-bool (val[e]) ≡ ¬(val-to-bool
(val[¬e]))
   by (smt (verit, best) Value.disc(1) Value.disc(10) Value.disc(4) Value.disc(5)
Value.disc(6) Value.disc(9) intval-logic-negation.elims val-to-bool.simps(1) val-to-bool.simps(2)
zero-neq-one)

optimization negate-condition: ((¬e) ? x : y) ⟼ (e ? y : x)
    apply unfold-optimization apply simp using negates
   using ConditionalExprE UnaryExprE intval-logic-negation.elims unary-eval.simps(4)
val-to-bool.simps(1) val-to-bool.simps(2) zero-neq-one
    apply (smt (verit) ConditionalExpr)
    unfolding size.simps by simp

optimization const-true: (true ? x : y) ⟼ x
   apply unfold-optimization
   apply force
  unfolding size.simps by simp

optimization const-false: (false ? x : y) ⟼ y
   apply unfold-optimization
   apply force
  unfolding size.simps by simp
```

optimization *equal-branches*: $(e\ ?\ x : x) \longmapsto x$
  apply *unfold-optimization*
  apply *force*
 unfolding *size.simps* **by** *auto*

definition *wff-stamps* :: *bool* **where**
 $\textit{wff-stamps} = (\forall\, m\ p\ expr\ val\ .\ ([m,p] \vdash expr \mapsto val) \longrightarrow \textit{valid-value}\ val\ (\textit{stamp-expr}\ expr))$

definition *wf-stamp* :: *IRExpr* $\Rightarrow$ *bool* **where**
 $\textit{wf-stamp}\ e = (\forall\, m\ p\ v.\ ([m,\ p] \vdash e \mapsto v) \longrightarrow \textit{valid-value}\ v\ (\textit{stamp-expr}\ e))$

optimization *condition-bounds-x*: $((u < v)\ ?\ x : y) \longmapsto x$
  when $(\textit{stamp-under}\ (\textit{stamp-expr}\ u)\ (\textit{stamp-expr}\ v) \wedge \textit{wf-stamp}\ u \wedge \textit{wf-stamp}\ v)$
  apply *unfold-optimization*
 using *stamp-under-semantics*
 using *wf-stamp-def*
 apply (*smt* (*verit*, *best*) *ConditionalExprE le-expr-def stamp-under.simps*)
 unfolding *size.simps* **by** *simp*

optimization *condition-bounds-y*: $((x < y)\ ?\ x : y) \longmapsto y$
  when $(\textit{stamp-under}\ (\textit{stamp-expr}\ y)\ (\textit{stamp-expr}\ x) \wedge \textit{wf-stamp}\ x \wedge \textit{wf-stamp}\ y)$
  apply *unfold-optimization*
 using *stamp-under-semantics-inversed*
 using *wf-stamp-def*
 apply (*smt* (*verit*, *best*) *ConditionalExprE le-expr-def stamp-under.simps*)
 unfolding *size.simps* **by** *simp*

optimization *b*[*intval*]: $((x\ eq\ y)\ ?\ x : y) \longmapsto y$
  apply *unfold-optimization*
   apply (*smt* (*z3*) *bool-to-val.simps*(*2*) *intval-equals.elims val-to-bool.simps*(*1*) *val-to-bool.simps*(*3*))
   unfolding *intval.simps*
  apply (*smt* (*z3*) *BinaryExprE ConditionalExprE Value.inject*(*1*) *Value.inject*(*2*) *bin-eval.simps*(*10*) *bool-to-val.simps*(*2*) *evalDet intval-equals.simps*(*1*) *intval-equals.simps*(*10*) *intval-equals.simps*(*12*) *intval-equals.simps*(*15*) *intval-equals.simps*(*16*) *intval-equals.simps*(*2*) *intval-equals.simps*(*5*) *intval-equals.simps*(*8*) *intval-equals.simps*(*9*) *le-expr-def val-to-bool.cases val-to-bool.elims*(*2*))
 unfolding *size.simps* **by** *auto*

end

end