Veriopt Theories

August 31, 2022

Contents

1 Canonicalization Phase

theory *Common*
  imports
    OptimizationDSL.Canonicalization
    Semantics.IRTreeEvalThms
begin

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*: $op \neq BinAdd \Longrightarrow size\ (BinaryExpr\ op\ a\ b) = size\ a + size\ b$
  by (*induction op*; *auto*)

lemma *size-non-const*:
  $\neg\ is\text{-}ConstantExpr\ y \Longrightarrow 1 < size\ y$
  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

definition *well-formed-equal* :: *Value* ⇒ *Value* ⇒ *bool*
(**infix** ≈ *50*) **where**
well-formed-equal v_1 v_2 = (v_1 ≠ *UndefVal* ⟶ v_1 = v_2)

lemma *well-formed-equal-defn* [*simp*]:
well-formed-equal v_1 v_2 = (v_1 ≠ *UndefVal* ⟶ v_1 = v_2)
unfolding *well-formed-equal-def* **by** *simp*

end

1.1 Conditional Expression

theory *ConditionalPhase*
imports
Common
begin

phase *Conditional*
terminating *size*
begin

lemma *negates*: *is-IntVal e* ⟹ *val-to-bool* (*val*[*e*]) ≡ ¬(*val-to-bool* (*val*[!*e*]))
using *intval-logic-negation.simps* **unfolding** *logic-negate-def*
sorry

lemma *negation-condition-intval*:

assumes *e* = *IntVal b ie*
assumes *0* < *b*
shows *val*[(!*e*) *?* *x* : *y*] = *val*[*e* *?* *y* : *x*]
using *assms* **by** (*cases e*; *auto simp*: *negates logic-negate-def*)

optimization *negate-condition*: ((!*e*) *?* *x* : *y*) ⟼ (*e* *?* *y* : *x*)
apply *simp* **using** *negation-condition-intval*
by (*smt* (*verit*, *ccfv-SIG*) *ConditionalExpr ConditionalExprE Value.collapse Value.exhaust-disc evaltree-not-undef intval-logic-negation.simps*(*4*) *intval-logic-negation.simps negates unary-eval.simps*(*4*) *unfold-unary*)

definition *wff-stamps* :: *bool* **where**
wff-stamps = (∀ *m p expr val* . ([*m*,*p*] ⊢ *expr* ↦ *val*) ⟶ *valid-value val* (*stamp-expr expr*))

definition *wf-stamp* :: *IRExpr* ⇒ *bool* **where**
wf-stamp e = (∀ *m p v*. ([*m*, *p*] ⊢ *e* ↦ *v*) ⟶ *valid-value v* (*stamp-expr e*))

```
optimization b[intval]: ((x eq y) ? x : y) ⟼ y
 sorry


lemma val-optimise-integer-test:
  assumes is-IntVal32 x
  shows intval-conditional (intval-equals val[(x & (IntVal32 1))] (IntVal32 0))
         (IntVal32 0) (IntVal32 1) =
         val[x & IntVal32 1]
   apply simp-all
  apply auto
 using bool-to-val.elims intval-equals.elims val-to-bool.simps(1) val-to-bool.simps(3)
  sorry

optimization val-conditional-eliminate-known-less: ((x < y) ? x : y) ⟼ x
                              when (stamp-under (stamp-expr x) (stamp-expr y)
                                   ∧ wf-stamp x ∧ wf-stamp y)
      apply auto
    using stamp-under.simps wf-stamp-def val-to-bool.simps
    sorry


optimization opt-conditional-eq-is-RHS: ((BinaryExpr BinIntegerEquals x y) ? x
: y) ⟼ y
   apply simp-all apply auto using b Canonicalization.intval.simps(1) evalDet
         intval-conditional.simps
  by (metis (mono-tags, lifting) evaltree-not-undef)


optimization opt-normalize-x: ((x eq const (IntVal 32 0)) ?
                              (const (IntVal 32 0)) : (const (IntVal 32 1))) ⟼ x
                         when (x = ConstantExpr (IntVal 32 0) | (x = ConstantExpr
(IntVal 32 1)))
  done


optimization opt-normalize-x2: ((x eq (const (IntVal 32 1))) ?
                               (const (IntVal 32 1)) : (const (IntVal 32 0))) ⟼ x
                                       when (x = ConstantExpr (IntVal 32 0) | (x =
ConstantExpr (IntVal 32 1)))
  done
```

optimization *opt-flip-x*: $((x\ eq\ (const\ (IntVal\ 32\ 0)))\ ?$
$(const\ (IntVal\ 32\ 1))\ :\ (const\ (IntVal\ 32\ 0)))\longmapsto$
$x \oplus (const\ (IntVal\ 32\ 1))$
when $(x = ConstantExpr\ (IntVal\ 32\ 0)\ |\ (x = ConstantExpr$
$(IntVal\ 32\ 1)))$
done

optimization *opt-flip-x2*: $((x\ eq\ (const\ (IntVal\ 32\ 1)))\ ?$
$(const\ (IntVal\ 32\ 0))\ :\ (const\ (IntVal\ 32\ 1)))\longmapsto$
$x \oplus (const\ (IntVal\ 32\ 1))$
when $(x = ConstantExpr\ (IntVal\ 32\ 0)\ |\ (x = ConstantExpr$
$(IntVal\ 32\ 1)))$
done

optimization *opt-optimise-integer-test*:
$(((x\ \&\ (const\ (IntVal\ 32\ 1)))\ eq\ (const\ (IntVal\ 32\ 0)))\ ?$
$(const\ (IntVal\ 32\ 0))\ :\ (const\ (IntVal\ 32\ 1)))\longmapsto$
$x\ \&\ (const\ (IntVal\ 32\ 1))$
when $(stamp\text{-}expr\ x = default\text{-}stamp)$
apply *simp-all*
apply *auto*
using *val-optimise-integer-test* **sorry**

optimization *opt-optimise-integer-test-2*:
$(((x\ \&\ (const\ (IntVal\ 32\ 1)))\ eq\ (const\ (IntVal\ 32\ 0)))\ ?$
$(const\ (IntVal\ 32\ 0))\ :\ (const\ (IntVal\ 32\ 1)))\longmapsto$
x
when $(x = ConstantExpr\ (IntVal\ 32\ 0)\ |\ (x = ConstantExpr\ (IntVal$
$32\ 1)))$
done

optimization *opt-conditional-eliminate-known-less*: $((x < y)\ ?\ x : y)\longmapsto x$
when $(((stamp\text{-}under\ (stamp\text{-}expr\ x)\ (stamp\text{-}expr\ y))\ |$
$((stpi\text{-}upper\ (stamp\text{-}expr\ x)) = (stpi\text{-}lower\ (stamp\text{-}expr$
$y))))$
$\wedge\ wf\text{-}stamp\ x \wedge wf\text{-}stamp\ y)$
unfolding *le-expr-def* **apply** *auto*
using *stamp-under.simps wf-stamp-def val-conditional-eliminate-known-less*
sorry

end

end