

# Veriopt Theories

September 6, 2022

## Contents

<b>1 Canonicalization Phase</b>	<b>1</b>
1.1 Conditional Expression . . . . .	2

## 1 Canonicalization Phase

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

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

lemma size-non-add:  $op \neq \text{BinAdd} \implies \text{size } (\text{BinaryExpr } op \ a \ b) = \text{size } a + \text{size } b$ 
  by (induction op; auto)

lemma size-non-const:
   $\neg \text{is-ConstantExpr } y \implies 1 < \text{size } y$ 
  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*  $\Rightarrow$  *Value*  $\Rightarrow$  *bool*  
 (*infix*  $\approx 50$ ) **where**  
*well-formed-equal* *v*<sub>1</sub> *v*<sub>2</sub> = (*v*<sub>1</sub>  $\neq$  *UndefVal*  $\longrightarrow$  *v*<sub>1</sub> = *v*<sub>2</sub>)

**lemma** *well-formed-equal-defn* [*simp*]:  
*well-formed-equal* *v*<sub>1</sub> *v*<sub>2</sub> = (*v*<sub>1</sub>  $\neq$  *UndefVal*  $\longrightarrow$  *v*<sub>1</sub> = *v*<sub>2</sub>)  
**unfolding** *well-formed-equal-def* **by** *simp*

**end**

## 1.1 Conditional Expression

**theory** *ConditionalPhase*  
**imports**  
*Common*  
**begin**

**phase** *ConditionalNode*  
**terminating** *size*  
**begin**

**lemma** *negates*: *is-IntVal e*  $\implies$  *val-to-bool* (*val*[*e*])  $\equiv \neg$ (*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** *NegateConditionFlipBranches*: (!*e*) ? *x* : *y*  $\mapsto$  (*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*)

**optimization** *DefaultTrueBranch*: (*true* ? *x* : *y*)  $\mapsto$  *x* .

**optimization** *DefaultFalseBranch*: (*false* ? *x* : *y*)  $\mapsto$  *y* .

**optimization** *ConditionalEqualBranches*: (*e* ? *x* : *x*)  $\mapsto$  *x* .

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

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

**lemma** *val-optimize-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** *ConditionalEliminateKnownLess*: ( $(x < y) ? x : y$ )  $\mapsto$  *x*  
*when* (*stamp-under* (*stamp-expr* *x*) (*stamp-expr* *y*)  
 $\wedge$  *wf-stamp* *x*  $\wedge$  *wf-stamp* *y*)  
**apply** *auto*  
**using** *stamp-under.simps* *wf-stamp-def* *val-to-bool.simps*  
**sorry**

**optimization** *ConditionalEqualIsRHS*: ( $(x \text{ eq } y) ? x : y$ )  $\mapsto$  *y*  
**apply** *simp-all* **apply** *auto* **using** *Canonicalization.intval.simps*(1) *evalDet*  
*intval-conditional.simps* *evaltree-not-undef*  
**by** (*metis* (*no-types*, *opaque-lifting*) *Value.discI*(2) *Value.distinct*(1) *intval-and.simps*(3)  
*intval-equals.simps*(2) *val-optimize-integer-test* *val-to-bool.simps*(2))

**optimization** *normalizeX*: ( $(x \text{ eq } \text{const } (\text{IntVal } 32\ 0)) ?$   
(*const* (*IntVal* 32 0)) : (*const* (*IntVal* 32 1)))  $\mapsto$  *x*  
*when* ( $x = \text{ConstantExpr } (\text{IntVal } 32\ 0) \mid x = \text{ConstantExpr}$   
(*IntVal* 32 1)))  
**done**

**optimization** *normalizeX2*: ( $(x \text{ eq } (\text{const } (\text{IntVal } 32\ 1))) ?$   
(*const* (*IntVal* 32 1)) : (*const* (*IntVal* 32 0)))  $\mapsto$  *x*  
*when* ( $x = \text{ConstantExpr } (\text{IntVal } 32\ 0) \mid x =$   
*ConstantExpr* (*IntVal* 32 1)))  
**done**

**optimization** *flipX*:  $((x \text{ eq } (\text{const } (\text{IntVal } 32 \ 0)))) \ ?$   
 $(\text{const } (\text{IntVal } 32 \ 1)) : (\text{const } (\text{IntVal } 32 \ 0))) \mapsto$   
 $x \oplus (\text{const } (\text{IntVal } 32 \ 1))$   
 $\text{when } (x = \text{ConstantExpr } (\text{IntVal } 32 \ 0) \mid (x = \text{ConstantExpr}$   
 $(\text{IntVal } 32 \ 1))))$   
**done**

**optimization** *flipX2*:  $((x \text{ eq } (\text{const } (\text{IntVal } 32 \ 1)))) \ ?$   
 $(\text{const } (\text{IntVal } 32 \ 0)) : (\text{const } (\text{IntVal } 32 \ 1))) \mapsto$   
 $x \oplus (\text{const } (\text{IntVal } 32 \ 1))$   
 $\text{when } (x = \text{ConstantExpr } (\text{IntVal } 32 \ 0) \mid (x = \text{ConstantExpr}$   
 $(\text{IntVal } 32 \ 1))))$   
**done**

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

**optimization** *opt-optimize-integer-test-2*:  
 $((((x \ \& \ (\text{const } (\text{IntVal } 32 \ 1))) \text{ eq } (\text{const } (\text{IntVal } 32 \ 0)))) \ ?$   
 $(\text{const } (\text{IntVal } 32 \ 0)) : (\text{const } (\text{IntVal } 32 \ 1))) \mapsto$   
 $x$   
 $\text{when } (x = \text{ConstantExpr } (\text{IntVal } 32 \ 0) \mid (x = \text{ConstantExpr } (\text{IntVal}$   
 $32 \ 1))))$   
**done**

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

**end**

end