

# Veriopt Theories

March 18, 2022

## Contents

<b>1</b>	<b>Optization DSLs</b>	<b>1</b>
1.1	Canonicalization DSL . . . . .	3

## 1 Optization DSLs

```
theory Markup
  imports Semantics.IRTreeEval Snippets.Snipping
begin
```

```
datatype 'a Rewrite =
  Transform 'a 'a (-  $\mapsto$  - 10) |
  Conditional 'a 'a bool (-  $\mapsto$  - when - 70) |
  Sequential 'a Rewrite 'a Rewrite |
  Transitive 'a Rewrite
```

```
datatype 'a ExtraNotation =
  ConditionalNotation 'a 'a 'a (- ? - : -) |
  EqualsNotation 'a 'a (- eq -) |
  ConstantNotation 'a (const - 120) |
  TrueNotation (true) |
  FalseNotation (false) |
  ExclusiveOr 'a 'a (-  $\oplus$  -)
```

**ML-file**  $\langle markup.ML \rangle$

```
ML <
structure IRExpTranslator : DSL-TRANSLATION =
struct
fun markup DSL-Tokens.Add = @{term BinaryExpr} $ @{term BinAdd}
  | markup DSL-Tokens.Sub = @{term BinaryExpr} $ @{term BinSub}
  | markup DSL-Tokens.Mul = @{term BinaryExpr} $ @{term BinMul}
  | markup DSL-Tokens.And = @{term BinaryExpr} $ @{term BinAnd}
  | markup DSL-Tokens.Or = @{term BinaryExpr} $ @{term BinOr}
  | markup DSL-Tokens.Xor = @{term BinaryExpr} $ @{term BinXor}
  | markup DSL-Tokens.Abs = @{term UnaryExpr} $ @{term UnaryAbs}
```

```

| markup DSL-Tokens.Less = @{term BinaryExpr} $ @{term BinIntegerLessThan}
| markup DSL-Tokens.Equals = @{term BinaryExpr} $ @{term BinIntegerEquals}
| markup DSL-Tokens.Not = @{term UnaryExpr} $ @{term UnaryLogicNegation}
| markup DSL-Tokens.Negate = @{term UnaryExpr} $ @{term UnaryNeg}
| markup DSL-Tokens.LeftShift = @{term BinaryExpr} $ @{term BinLeftShift}
| markup DSL-Tokens.RightShift = @{term BinaryExpr} $ @{term BinRightShift}
| markup DSL-Tokens.UnsignedRightShift = @{term BinaryExpr} $ @{term Bin-
URightShift}
| markup DSL-Tokens.Conditional = @{term ConditionalExpr}
| markup DSL-Tokens.Constant = @{term ConstantExpr}
| markup DSL-Tokens.TrueConstant = @{term ConstantExpr (IntVal32 1)}
| markup DSL-Tokens.FalseConstant = @{term ConstantExpr (IntVal32 0)}
end

```

```

structure IntValTranslator : DSL-TRANSLATION =
struct

```

```

fun markup DSL-Tokens.Add = @{term intval-add}
| markup DSL-Tokens.Sub = @{term intval-sub}
| markup DSL-Tokens.Mul = @{term intval-mul}
| markup DSL-Tokens.And = @{term intval-and}
| markup DSL-Tokens.Or = @{term intval-or}
| markup DSL-Tokens.Xor = @{term intval-xor}
| markup DSL-Tokens.Abs = @{term intval-abs}
| markup DSL-Tokens.Less = @{term intval-less-than}
| markup DSL-Tokens.Equals = @{term intval-equals}
| markup DSL-Tokens.Not = @{term intval-logic-negation}
| markup DSL-Tokens.Negate = @{term intval-negate}
| markup DSL-Tokens.LeftShift = @{term intval-left-shift}
| markup DSL-Tokens.RightShift = @{term intval-right-shift}
| markup DSL-Tokens.UnsignedRightShift = @{term intval-uright-shift}
| markup DSL-Tokens.Conditional = @{term intval-conditional}
| markup DSL-Tokens.Constant = @{term IntVal32}
| markup DSL-Tokens.TrueConstant = @{term IntVal32 1}
| markup DSL-Tokens.FalseConstant = @{term IntVal32 0}
end

```

```

structure IRExprMarkup = DSL-Markup(IRExprTranslator);
structure IntValMarkup = DSL-Markup(IntValTranslator);
>

```

*ir expression translation*

```

syntax -expandExpr :: term ⇒ term (exp[-])
parse-translation < [( @{syntax-const -expandExpr} , IREx-
prMarkup.markup-expr []) >

```

*value expression translation*

```
syntax -expandIntVal :: term ⇒ term (val[-])
parse-translation < [( @{syntax-const -expandIntVal} , IntVal-
Markup.markup-expr []) ] >
```

*ir expression example*

```
value exp[(e1 < e2) ? e1 : e2]

ConditionalExpr (BinaryExpr BinIntegerLessThan e1 e2) e1 e2
```

*value expression example*

```
value val[(e1 < e2) ? e1 : e2]

intval-conditional (intval-less-than e1 e2) e1 e2
```

```
value exp[((e1 - e2) + (const (IntVal32 0)) + e2) ⟶ e1 when True]
value val[((e1 - e2) + (const 0) + e2) ⟶ e1 when True]
```

```
end
theory Phase
  imports Main
begin
```

```
ML-file map.ML
ML-file phase.ML
```

```
end
```

## 1.1 Canonicalization DSL

```
theory Canonicalization
  imports
    Markup
    Phase
  keywords
    phase :: thy-decl and
    terminating :: quasi-command and
    print-phases :: diag and
    optimization :: thy-goal-defn
begin
```

```
ML <
datatype 'a Rewrite =
  Transform of 'a * 'a |
  Conditional of 'a * 'a * term |
```

```

    Sequential of 'a Rewrite * 'a Rewrite |
    Transitive of 'a Rewrite

type rewrite = {name: string, rewrite: term Rewrite}

structure RewriteRule : Rule =
struct
type T = rewrite;

fun pretty-rewrite ctxt (Transform (from, to)) =
    Pretty.block [
        Syntax.pretty-term ctxt from,
        Pretty.str  $\mapsto$  ,
        Syntax.pretty-term ctxt to
    ]
| pretty-rewrite ctxt (Conditional (from, to, cond)) =
    Pretty.block [
        Syntax.pretty-term ctxt from,
        Pretty.str  $\mapsto$  ,
        Syntax.pretty-term ctxt to,
        Pretty.str when ,
        Syntax.pretty-term ctxt cond
    ]
| pretty-rewrite - - = Pretty.str not implemented

fun pretty ctxt t =
    Pretty.block [
        Pretty.str ((#name t) ^ ":"),
        pretty-rewrite ctxt (#rewrite t)
    ]
end

structure RewritePhase = DSL-Phase(RewriteRule);

val - =
    Outer-Syntax.command command-keyword <phase> enter an optimization phase
    (Parse.binding --| Parse.$$$ terminating -- Parse.const --| Parse.begin
    >> (Toplevel.begin-main-target true o RewritePhase.setup));

fun print-phases ctxt =
    let
        val thy = Proof-Context.theory-of ctxt;
        fun print phase = RewritePhase.pretty phase ctxt
    in
        map print (RewritePhase.phases thy)
    end

fun print-optimizations thy =
    print-phases thy |> Pretty.writeln-chunks

```

```

val - =
  Outer-Syntax.command command-keyword ⟨print-phases⟩
    print debug information for optimizations
    (Scan.succeed
      (Toplevel.keep (print-optimizations o Toplevel.context-of)));
  >

```

**ML-file** *rewrites.ML*

```

fun rewrite-preservation :: IRExp Rewrite ⇒ bool where
  rewrite-preservation (Transform x y) = (y ≤ x) |
  rewrite-preservation (Conditional x y cond) = (cond ⟶ (y ≤ x)) |
  rewrite-preservation (Sequential x y) = (rewrite-preservation x ∧ rewrite-preservation
y) |
  rewrite-preservation (Transitive x) = rewrite-preservation x

fun rewrite-termination :: IRExp Rewrite ⇒ (IRExp ⇒ nat) ⇒ bool where
  rewrite-termination (Transform x y) trm = (trm x > trm y) |
  rewrite-termination (Conditional x y cond) trm = (cond ⟶ (trm x > trm y)) |
  rewrite-termination (Sequential x y) trm = (rewrite-termination x trm ∧ rewrite-termination
y trm) |
  rewrite-termination (Transitive x) trm = rewrite-termination x trm

fun intval :: Value Rewrite ⇒ bool where
  intval (Transform x y) = (x ≠ UndefVal ∧ y ≠ UndefVal ⟶ x = y) |
  intval (Conditional x y cond) = (cond ⟶ (x = y)) |
  intval (Sequential x y) = (intval x ∧ intval y) |
  intval (Transitive x) = intval x

```

```

ML <
  structure System : RewriteSystem =
  struct
    val preservation = @{const rewrite-preservation};
    val termination = @{const rewrite-termination};
    val intval = @{const intval};
  end

```

```

  structure DSL = DSL-Rewrites(System);

```

```

val - =
  Outer-Syntax.local-theory-to-proof command-keyword ⟨optimization⟩
    define an optimization and open proof obligation
    (Parse-Spec.thm-name : -- Parse.term
      >> DSL.rewrite-cmd);
  >

```

**end**