Veriopt Theories

September 13, 2022

Contents

1   Optization DSLs

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

datatype *'a Rewrite =*
  Transform 'a 'a (- ⟼ - 10) |
  Conditional 'a 'a bool (- ⟼ - 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 (- ⊕ -) |
  LogicNegationNotation 'a (!-) |
  ShortCircuitOr 'a 'a (- || -)

definition *word :: ('a::len) word ⇒ 'a word* **where**
  word x = x

ML-file *‹markup.ML›*

ML *‹*
structure IRExprTranslator : 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.ShortCircuitOr = @{term BinaryExpr} $ @{term Bin-
ShortCircuitOr}
    | 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 UnaryNot}
    | markup DSL-Tokens.Negate = @{term UnaryExpr} $ @{term UnaryNeg}
    | markup DSL-Tokens.LogicNegate = @{term UnaryExpr} $ @{term UnaryLog-
icNegation}
    | markup DSL-Tokens.LeftShift = @{term BinaryExpr} $ @{term BinLeftShift}
    | markup DSL-Tokens.RightShift = @{term BinaryExpr} $ @{term BinRight-
Shift}
    | 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 (IntVal 32 1)}
    | markup DSL-Tokens.FalseConstant = @{term ConstantExpr (IntVal 32 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.ShortCircuitOr = @{term intval-short-circuit-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-not}
    | markup DSL-Tokens.Negate = @{term intval-negate}
    | markup DSL-Tokens.LogicNegate = @{term intval-logic-negation}
    | 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 IntVal 32}
    | markup DSL-Tokens.TrueConstant = @{term IntVal 32 1}
    | markup DSL-Tokens.FalseConstant = @{term IntVal 32 0}
end

structure WordTranslator : DSL-TRANSLATION =
```

struct
fun markup DSL-Tokens.Add = @{term plus}
  | markup DSL-Tokens.Sub = @{term minus}
  | markup DSL-Tokens.Mul = @{term times}
 | markup DSL-Tokens.And = @{term Bit-Operations.semiring-bit-operations-class.and}
  | markup DSL-Tokens.Or = @{term or}
  | markup DSL-Tokens.Xor = @{term xor}
  | markup DSL-Tokens.Abs = @{term abs}
  | markup DSL-Tokens.Less = @{term less}
  | markup DSL-Tokens.Equals = @{term HOL.eq}
  | markup DSL-Tokens.Not = @{term not}
  | markup DSL-Tokens.Negate = @{term uminus}
  | markup DSL-Tokens.LogicNegate = @{term logic-negate}
  | markup DSL-Tokens.LeftShift = @{term shiftl}
  | markup DSL-Tokens.RightShift = @{term signed-shiftr}
  | markup DSL-Tokens.UnsignedRightShift = @{term shiftr}
  | markup DSL-Tokens.Constant = @{term word}
  | markup DSL-Tokens.TrueConstant = @{term 1}
  | markup DSL-Tokens.FalseConstant = @{term 0}
end

structure IRExprMarkup = DSL-Markup(IRExprTranslator);
structure IntValMarkup = DSL-Markup(IntValTranslator);
structure WordMarkup = DSL-Markup(WordTranslator);
›

---

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* [])] ›

---

word expression translation

syntax *-expandWord :: term ⇒ term (bin[-])*
parse-translation ‹ [(@{*syntax-const -expandWord*} , *Word-Markup.markup-expr* [])] ›

value $exp[((e_1 - e_2) + (const\ (IntVal\ 32\ 0)) + e_2) \longmapsto e_1\ when\ True]$

value $bin[-x]$
value $val[-x]$
value $exp[-x]$

value $bin[!x]$
value $val[!x]$
value $exp[!x]$

value $bin[\neg x]$
value $val[\neg x]$
value $exp[\neg x]$

value $bin[{\sim}x]$
value $val[{\sim}x]$
value $exp[{\sim}x]$

value ${\sim}x$

end
theory *Phase*
  imports *Main*
begin

ML-file *map.ML*
ML-file *phase.ML*

end

4

1.1 Canonicalization DSL

theory *Canonicalization*
  imports
    Markup
    Phase
    HOL−Eisbach.Eisbach
  keywords
    phase :: *thy-decl* **and**
    terminating :: *quasi-command* **and**
    print-phases :: *diag* **and**
    export-phases :: *thy-decl* **and**
    optimization :: *thy-goal-defn*
begin

print-methods

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: binding,
  rewrite: term Rewrite,
  proofs: thm list,
  code: thm list,
  source: term
}

structure RewriteRule : Rule =
struct
type T = rewrite;

(*
fun pretty-rewrite ctxt (Transform (from, to)) =
      Pretty.block [
        Syntax.pretty-term ctxt from,
        Pretty.str  ↦ ,
        Syntax.pretty-term ctxt to
      ]
  | pretty-rewrite ctxt (Conditional (from, to, cond)) =
      Pretty.block [
        Syntax.pretty-term ctxt from,
        Pretty.str  ↦ ,
        Syntax.pretty-term ctxt to,
        Pretty.str  when ,
        Syntax.pretty-term ctxt cond
```

```
        ]
    | pretty-rewrite - - = Pretty.str not implemented*)

fun pretty-thm ctxt thm =
  (Proof-Context.pretty-fact ctxt (, [thm]))

fun pretty ctxt obligations t =
  let
    val is-skipped = Thm-Deps.has-skip-proof (#proofs t);

    val warning = (if is-skipped
      then [Pretty.str (proof skipped), Pretty.brk 0]
      else []);

    val obligations = (if obligations
      then [Pretty.big-list
            obligations:
            (map (pretty-thm ctxt) (#proofs t)),
          Pretty.brk 0]
      else []);

    fun pretty-bind binding =
      Pretty.markup
        (Position.markup (Binding.pos-of binding) Markup.position)
        [Pretty.str (Binding.name-of binding)];

  in
  Pretty.block ([
    pretty-bind (#name t), Pretty.str : ,
    Syntax.pretty-term ctxt (#source t), Pretty.fbrk
  ] @ obligations @ warning)
  end
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 print-obligations ctxt =
  let
    val thy = Proof-Context.theory-of ctxt;
    fun print phase = RewritePhase.pretty print-obligations phase ctxt
  in
    map print (RewritePhase.phases thy)
  end
```

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

val - =
  Outer-Syntax.command command-keyword ‹print-phases›
    print debug information for optimizations
    (Parse.opt-bang >>
      (fn b => Toplevel.keep ((print-optimizations b) o Toplevel.context-of)));

fun export-phases thy name =
  let
    val state = Toplevel.theory-toplevel thy;
    val ctxt = Toplevel.context-of state;
    val content = Pretty.string-of (Pretty.chunks (print-phases false ctxt));
    val cleaned = YXML.content-of content;


    val filename = Path.explode (name^.rules);
    val directory = Path.explode optimizations;
    val path = Path.binding (
                Path.append directory filename,
                Position.none);
    val thy' = thy |> Generated-Files.add-files (path, content);

    val - = Export.export thy' path [YXML.parse cleaned];

    val - = writeln (Export.message thy' (Path.basic optimizations));
  in
    thy'
  end

val - =
  Outer-Syntax.command command-keyword ‹export-phases›
    export information about encoded optimizations
    (Parse.text >>
      (fn name => Toplevel.theory (fn state => export-phases state name)))
›

ML-file rewrites.ML

phase Opt
  terminating size
begin

end

print-phases
export-phases ‹MyPhases›
```

fun *rewrite-preservation* :: *IRExpr 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* :: *IRExpr Rewrite* ⇒ (*IRExpr* ⇒ *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*

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*

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

method *unfold-size* =
  (*unfold size.simps*, *simp del*: *le-expr-def*)*?*
  | (*unfold size.simps*)*?*

print-methods

ML ‹
structure System : *RewriteSystem* =
struct
val preservation = @{*const rewrite-preservation*};
val termination = @{*const rewrite-termination*};

8

```
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
```