

Veriopt Theories

August 25, 2022

Contents

theory *TreeSnippets*
imports
 Canonicalizations.ConditionalPhase
 Optimizations.CanonicalizationSyntax
 Semantics.TreeToGraphThms
 Snippets.Snipping
 HOL-Library.OptionalSugar
begin

no-notation *ConditionalExpr* (- ? - : -)

notation (*latex*)
 kind (-⟨-⟩)

notation (*latex*)
 valid-value (- ∈ -)

notation (*latex*)
 val-to-bool (*bool-of* -)

notation (*latex*)
 constantAsStamp (*stamp-from-value* -)

notation (*latex*)
 size (*trm*(-))

translations
 $y > x \leq x < y$

notation (*latex*)
 greater (- > / -)

translations

$n \leq \text{CONST Rep-intexp } n$
 $n \leq \text{CONST Rep-i32exp } n$
 $n \leq \text{CONST Rep-i64exp } n$

lemma *vminusv*: $\forall vv \ v . vv = \text{IntVal32 } v \longrightarrow v - v = 0$
by *simp*
thm-oracles *vminusv*

lemma *vminusv2*: $\forall v::\text{int32} . v - v = 0$
by *simp*

lemma *redundant-sub*:
 $\forall vv_1 \ vv_2 \ v_1 \ v_2 . vv_1 = \text{IntVal32 } v_1 \wedge vv_2 = \text{IntVal32 } v_2 \longrightarrow v_1 - (v_1 - v_2) = v_2$
by *simp*
thm-oracles *redundant-sub*

lemma *redundant-sub2*:
 $\forall (v_1::\text{int32}) (v_2::\text{int32}) . v_1 - (v_1 - v_2) = v_2$
by *simp*

val-eq

$\forall (vv::\text{Value}) \ v :: 32 \ \text{word} . vv = \text{IntVal32 } v \longrightarrow v - v = (0 :: 32 \ \text{word})$

$\forall (vv_1::\text{Value}) (vv_2::\text{Value}) (v_1::32 \ \text{word}) \ v_2 :: 32 \ \text{word} . vv_1 = \text{IntVal32 } v_1 \wedge vv_2 = \text{IntVal32 } v_2 \longrightarrow v_1 - (v_1 - v_2) = v_2$

lemma *sub-same-32-val*:
assumes $\text{val}[e - e] \neq \text{UndefVal}$
assumes $\text{is-IntVal32 } e$
shows $\text{val}[e - e] = \text{val}[\text{const } 0]$
using *assms* **by** (*cases e; auto*)

phase *tmp*
terminating *size*
begin

sub-same-32

optimization *sub-same-32*: $(e::i32exp) - e \longmapsto \text{const } (\text{IntVal32 } 0)$

defer apply *simp* **using** *sub-same-32-val*
apply (*metis Value.disc(2) bin-eval.simps(3) evalDet i32e-eval unfold-binary unfold-const32*)
unfolding *size.simps*
by (*metis add-strict-increasing gr-implies-not0 less-one linorder-not-le size-gt-0*)

```

lemma sub-same-64-val:
  assumes  $\text{val}[e - e] \neq \text{UndefVal}$ 
  assumes is-IntVal64  $e$ 
  shows  $\text{val}[e - e] = \text{val}[\text{IntVal64 } 0]$ 
  using assms by (cases  $e$ ; auto)

```

sub-same-64

optimization *sub-same-64*: $(e :: i64\text{exp}) - e \mapsto \text{const } (\text{IntVal64 } 0)$

```

defer apply simp using sub-same-64-val
apply (metis Value.discI(2) bin-eval.simps(3) evalDet i64e-eval unfold-binary
unfold-const64)
by (simp add: Suc-le-eq add-strict-increasing size-gt-0)
end

```

thm-oracles *sub-same-32*

ast-example

```

BinaryExpr BinAdd (BinaryExpr BinMul ( $x :: \text{IRExpr}$ )  $x$ )
(BinaryExpr BinMul  $x$   $x$ )

```

abstract-syntax-tree

```

datatype IRExpr =
  UnaryExpr IRUnaryOp IRExpr
| BinaryExpr IRBinaryOp IRExpr IRExpr
| ConditionalExpr IRExpr IRExpr IRExpr
| ParameterExpr nat Stamp
| LeafExpr nat Stamp
| ConstantExpr Value
| ConstantVar (char list)
| VariableExpr (char list) Stamp

```

value

```

datatype Value = UndefVal
| IntVal32 (32 word)
| IntVal64 (64 word)
| ObjRef (nat option)
| ObjStr (char list)

```

eval

$\text{unary-eval} :: \text{IRUnaryOp} \Rightarrow \text{Value} \Rightarrow \text{Value}$

$\text{bin-eval} :: \text{IRBinaryOp} \Rightarrow \text{Value} \Rightarrow \text{Value} \Rightarrow \text{Value}$

tree-semantic

$\text{semantics:unary} \quad \text{semantics:binary} \quad \text{semantics:conditional} \quad \text{semantics:constant} \quad \text{semantics:parameter} \quad \text{semantics:leaf}$

tree-evaluation-deterministic

$[m :: \text{nat} \Rightarrow \text{Value}, p :: \text{Value list}] \vdash e :: \text{IRExpr} \mapsto v_1 :: \text{Value} \wedge$
 $[m, p] \vdash e \mapsto v_2 :: \text{Value} \implies$
 $v_1 = v_2$

thm-oracles *evalDet*

expression-refinement

$(e_1 :: \text{IRExpr}) \sqsubseteq (e_2 :: \text{IRExpr}) = (\forall (m :: \text{nat} \Rightarrow \text{Value}) (p :: \text{Value list})$
 $v :: \text{Value}. [m, p] \vdash e_1 \mapsto v \longrightarrow [m, p] \vdash e_2 \mapsto v)$

expression-refinement-monotone

$(e :: \text{IRExpr}) \sqsubseteq (e' :: \text{IRExpr})$
 $(x :: \text{IRExpr}) \sqsubseteq (x' :: \text{IRExpr}) \wedge (y :: \text{IRExpr}) \sqsubseteq (y' :: \text{IRExpr})$
 $(ce :: \text{IRExpr}) \sqsubseteq (ce' :: \text{IRExpr}) \wedge (te :: \text{IRExpr}) \sqsubseteq (te' :: \text{IRExpr}) \wedge (fe :: \text{IRExpr}) \sqsubseteq (fe' :: \text{IRExpr})$

ML \hookleftarrow

$(\text{*fun get-list (phase: phase option) =}$
 case phase of
 $\text{NONE} => [] \mid$
 $\text{SOME } p => (\#rewrites p)$

$\text{fun get-rewrite name thy =}$
 let
 $\text{val (phases, lookup) = (case RWList.get thy of}$
 $\text{NoPhase store} => \text{store} \mid$
 $\text{InPhase (name, store, -)} => \text{store})$
 $\text{val rewrites = (map (fn x => get-list (lookup x)) phases)}$
 in
 rewrites
 end

```

fun rule-print name =
  Document-Output.antiquotation-pretty name (Args.term)
  (fn ctxt => fn (rule) => (*Pretty.str hello*)
    Pretty.block (print-all-phases (Proof-Context.theory-of ctxt)));
(*)
  Goal-Display.pretty-goal
  (Config.put Goal-Display.show-main-goal main ctxt)
  (#goal (Proof.goal (Toplevel.proof-of (Toplevel.presentation-state ctxt))));
*)

val - = Theory.setup
  (rule-print binding <rule>);
>

```

phase *SnipPhase*
terminating *size*
begin

BinaryFoldConstant

optimization *BinaryFoldConstant*: *BinaryExpr op (const v1) (const v2)*
 \mapsto *ConstantExpr (bin-eval op v1 v2)* when *int-and-equal-bits v1 v2*

unfolding *rewrite-preservation.simps rewrite-termination.simps*
apply (rule *conjE*, *simp*, *simp del*: *le-expr-def*)

BinaryFoldConstantObligation

1. *int-and-equal-bits v1 v2* \longrightarrow
 $\text{trm}(\text{BinaryExpr op } (\text{ConstantExpr } v1) (\text{ConstantExpr } v2)) > \text{Suc } (0 :: \text{nat})$
2. *int-and-equal-bits v1 v2* \longrightarrow
 $\text{BinaryExpr op } (\text{ConstantExpr } v1) (\text{ConstantExpr } v2) \sqsubseteq$
 $\text{ConstantExpr (bin-eval op v1 v2)}$

variables:

op :: *IRBinaryOp*
v1, *v2* :: *Value*

using *BinaryFoldConstant* **by** *auto*

AddCommuteConstantRight

optimization *AddCommuteConstantRight*: $((\text{const } v) + y) \mapsto y + (\text{const } v)$ when $\neg(\text{is-ConstantExpr } y)$

unfolding *rewrite-preservation.simps rewrite-termination.simps*

apply (rule conjE, simp, simp del: le-expr-def)

AddCommuteConstantRightObligation

1. $\neg \text{is-ConstantExpr } y \longrightarrow \text{trm}(y) > \text{Suc } (0 :: \text{nat})$
2. $\neg \text{is-ConstantExpr } y \longrightarrow$
 $\text{BinaryExpr BinAdd (ConstantExpr } v) y \sqsubseteq$
 $\text{BinaryExpr BinAdd } y \text{ (ConstantExpr } v)$

variables:

$v :: \text{Value}$

$y :: \text{IRExpr}$

using AddShiftConstantRight **by** auto

AddNeutral

optimization AddNeutral: $((e :: i32exp) + (\text{const } (\text{IntVal32 } 0))) \longmapsto e$

unfolding rewrite-preservation.simps rewrite-termination.simps

apply (rule conjE, simp, simp del: le-expr-def)

AddNeutralObligation

1. $\text{BinaryExpr BinAdd } e \text{ (ConstantExpr (IntVal32 } (0 :: 32 \text{ word}))) \sqsubseteq e$

variables:

$e :: i32exp$

using neutral-zero(1) rewrite-preservation.simps(1) **by** blast

InverseLeftSub

optimization InverseLeftSub: $((e_1 :: interp) - (e_2 :: interp)) + e_2 \longmapsto e_1$

unfolding rewrite-preservation.simps rewrite-termination.simps

apply (rule conjE, simp, simp del: le-expr-def)

InverseLeftSubObligation

1. $\text{trm}(e_2) > 0 :: \text{nat}$
2. $\text{BinaryExpr BinAdd (BinaryExpr BinSub } e_1 \text{ } e_2) e_2 \sqsubseteq e_1$

variables:

$e_1, e_2 :: interp$

using neutral-left-add-sub **by** auto

InverseRightSub

optimization InverseRightSub: $(e_2 :: interp) + ((e_1 :: interp) - e_2) \longmapsto e_1$

unfolding *rewrite-preservation.simps rewrite-termination.simps*
apply (rule *conjE*, *simp*, *simp del: le-expr-def*)

InverseRightSubObligation

1. $\text{trm}(e_1) > 0 :: \text{nat} \vee \text{trm}(e_2) > 0 :: \text{nat}$
 2. $\text{BinaryExpr BinAdd } e_2 (\text{BinaryExpr BinSub } e_1 \ e_2) \sqsupseteq e_1$
- variables:
 $e_1, e_2 :: \text{intexp}$

using *neutral-right-add-sub* **by** *auto*

AddToSub

optimization *AddToSub*: $-e + y \mapsto y - e$

unfolding *rewrite-preservation.simps rewrite-termination.simps*
apply (rule *conjE*, *simp*, *simp del: le-expr-def*)

AddToSubObligation

1. $\text{BinaryExpr BinAdd } (\text{UnaryExpr UnaryNeg } e) \ y \sqsupseteq \text{BinaryExpr BinSub } y \ e$
- variables:
 $e, y :: \text{IRExpr}$

using *AddLeftNegateToSub* **by** *auto*

end

definition *trm* **where** *trm = size*

phase

phase *AddCanonicalizations*
terminating *trm*
begin...end

hide-const (**open**) *Form.wf-stamp*

phase-example

phase *Conditional*
terminating *trm*
begin

phase-example-1

optimization *negate-condition*: $((!e) \text{ ? } x : y) \mapsto (e \text{ ? } y : x)$

using *ConditionalPhase.negate-condition*
by (*auto simp: trm-def*)

phase-example-2

optimization *const-true*: $(\text{true} \text{ ? } x : y) \mapsto x$

by (*auto simp: trm-def*)

phase-example-3

optimization *const-false*: $(\text{false} \text{ ? } x : y) \mapsto y$

by (*auto simp: trm-def*)

phase-example-4

optimization *equal-branches*: $(e \text{ ? } x : x) \mapsto x$

by (*auto simp: trm-def*)

phase-example-7

end

termination

$\text{trm}(\text{UnaryExpr } (op :: \text{IRUnaryOp}) (e :: \text{IRExpr}))$	$= \text{trm}(e :: \text{IRExpr}) + (1 :: \text{nat})$
$\text{trm}(\text{BinaryExpr } \text{BinAdd } (x :: \text{IRExpr}) (y :: \text{IRExpr}))$	$= \text{trm}(x :: \text{IRExpr}) + (2 :: \text{nat})$
$\text{trm}(\text{BinaryExpr } \text{BinIntegerBelow } (x :: \text{IRExpr}) (y :: \text{IRExpr}))$	$= \text{trm}(x :: \text{IRExpr}) + \text{trm}(y :: \text{IRExpr})$
$\text{trm}(\text{ConditionalExpr } (cond :: \text{IRExpr}) (t :: \text{IRExpr}) (f :: \text{IRExpr}))$	$= \text{trm}(cond :: \text{IRExpr}) + \text{trm}(t :: \text{IRExpr}) + \text{trm}(f :: \text{IRExpr})$
$\text{trm}(\text{ConstantExpr } (c :: \text{Value}))$	$= 1 :: \text{nat}$
$\text{trm}(\text{ParameterExpr } (ind :: \text{nat}) (s :: \text{Stamp}))$	$= 2 :: \text{nat}$

graph-representation

typedef *IRGraph* = $\{g :: ID \rightarrow (\text{IRNode} \times \text{Stamp}) . \text{finite } (\text{dom } g)\}$

graph2tree

rep:constant rep:parameter rep:conditional rep:unary rep:convert
rep:binary rep:leaf rep:ref

preeval

is-preevaluated (*InvokeNode* (*n* :: *nat*) (*uu* :: *nat*) (*uv* :: *nat option*) (*uw* :: *nat option*) (*ux* :: *nat option*) (*uy* :: *nat*)) = *True*

is-preevaluated (*InvokeWithExceptionNode* (*n* :: *nat*) (*uz* :: *nat*) (*va* :: *nat option*) (*vb* :: *nat option*) (*vc* :: *nat option*) (*vd* :: *nat*) (*ve* :: *nat*)) = *True*

is-preevaluated (*NewInstanceNode* (*n* :: *nat*) (*vf* :: *char list*) (*vg* :: *nat option*) (*vh* :: *nat*)) = *True*

is-preevaluated (*LoadFieldNode* (*n* :: *nat*) (*vi* :: *char list*) (*vj* :: *nat option*) (*vk* :: *nat*)) = *True*

is-preevaluated (*SignedDivNode* (*n* :: *nat*) (*vl* :: *nat*) (*vm* :: *nat*) (*vn* :: *nat option*) (*vo* :: *nat option*) (*vp* :: *nat*)) = *True*

is-preevaluated (*SignedRemNode* (*n* :: *nat*) (*vq* :: *nat*) (*vr* :: *nat*) (*vs* :: *nat option*) (*vt* :: *nat option*) (*vu* :: *nat*)) = *True*

is-preevaluated (*ValuePhiNode* (*n* :: *nat*) (*vv* :: *nat list*) (*vw* :: *nat*)) = *True*

is-preevaluated (*AbsNode* (*v* :: *nat*)) = *False*

is-preevaluated (*AddNode* (*v* :: *nat*) (*va* :: *nat*)) = *False*

is-preevaluated (*AndNode* (*v* :: *nat*) (*va* :: *nat*)) = *False*

is-preevaluated (*BeginNode* (*v* :: *nat*)) = *False*

is-preevaluated (*BytecodeExceptionNode* (*v* :: *nat list*) (*va* :: *nat option*) (*vb* :: *nat*)) = *False*

is-preevaluated (*ConditionalNode* (*v* :: *nat*) (*va* :: *nat*) (*vb* :: *nat*)) = *False*

is-preevaluated (*ConstantNode* (*v* :: *Value*)) = *False*

is-preevaluated (*DynamicNewArrayNode* (*v* :: *nat*) (*va* :: *nat*) (*vb* :: *nat option*) (*vc* :: *nat option*) (*vd* :: *nat*)) = *False*

is-preevaluated *EndNode* = *False*

is-preevaluated (*ExceptionObjectNode* (*v* :: *nat option*) (*va* :: *nat*)) = *False*

is-preevaluated (*FrameState* (*v* :: *nat list*) (*va* :: *nat option*) (*vb* :: *nat list option*) (*vc* :: *nat list option*)) = *False*

is-preevaluated (*IfNode* (*v* :: *nat*) (*va* :: *nat*) (*vb* :: *nat*)) = *False*

is-preevaluated (*IntegerBelowNode* (*v* :: *nat*) (*va* :: *nat*)) = *False*

is-preevaluated (*IntegerEqualsNode* (*v* :: *nat*) (*va* :: *nat*)) = *False*

is-preevaluated (*IntegerLessThanNode* (*v* :: *nat*) (*va* :: *nat*)) = *False*

is-preevaluated (*IsNullNode* (*v* :: *nat*)) = *False*

is-preevaluated (*KillingBeginNode* (*v* :: *nat*)) = *False*

is-preevaluated (*LeftShiftNode* (*v* :: *nat*) (*va* :: *nat*)) = *False*

is-preevaluated (*LogicNegationNode* (*v* :: *nat*)) = *False*

is-preevaluated (*LoopBeginNode* (\emptyset :: *nat list*) (*va* :: *nat option*) (*vb* :: *nat option*) (*vc* :: *nat*)) = *False*

is-preevaluated (*LoopEndNode* (*v* :: *nat*)) = *False*

is-preevaluated (*LoopExitNode* (*v* :: *nat*) (*va* :: *nat option*) (*vb* :: *nat*)) = *False*

is-preevaluated (*MergeNode* (*v* :: *nat list*) (*va* :: *nat option*) (*vb* :: *nat*)) = *False*

deterministic-representation

$$g :: IRGraph \vdash n :: nat \simeq e_1 :: IRExpr \wedge g \vdash n \simeq e_2 :: IRExpr \implies e_1 = e_2$$

thm-oracles *repDet*

well-formed-term-graph

$$\exists e :: IRExpr. g :: IRGraph \vdash n :: nat \simeq e \wedge (\exists v :: Value. [m :: nat \Rightarrow Value, p :: Value list] \vdash e \mapsto v)$$

graph-semantics

$$([g :: IRGraph, m :: nat \Rightarrow Value, p :: Value list] \vdash n :: nat \mapsto v :: Value) = (\exists e :: IRExpr. g \vdash n \simeq e \wedge [m, p] \vdash e \mapsto v)$$

graph-semantics-deterministic

$$[g :: IRGraph, m :: nat \Rightarrow Value, p :: Value list] \vdash n :: nat \mapsto v_1 :: Value \wedge [g, m, p] \vdash n \mapsto v_2 :: Value \implies v_1 = v_2$$

thm-oracles *graphDet*

notation (*latex*)

graph-refinement (*term-graph-refinement* -)

graph-refinement

$$\begin{aligned} &term-graph-refinement\ g_1 :: IRGraph\ (g_2 :: IRGraph) = \\ &(\text{ids } g_1 \subseteq \text{ids } g_2 \wedge \\ &(\forall n :: nat. \\ &\quad n \in \text{ids } g_1 \longrightarrow \\ &\quad (\forall e :: IRExpr. g_1 \vdash n \simeq e \longrightarrow g_2 \vdash n \leq e))) \end{aligned}$$

translations

$n \leq CONST$ as-set n

graph-semantics-preservation

$$\begin{aligned}
& (e_1' :: IRExpr) \sqsupseteq \\
& (e_2' :: IRExpr) \wedge \\
& \{n :: nat\} \triangleleft g_1 :: IRGraph \\
& \subseteq (g_2 :: IRGraph) \wedge \\
& g_1 \vdash n \simeq e_1' \wedge g_2 \vdash n \simeq e_2' \implies \\
& \text{term-graph-refinement } g_1 \ g_2
\end{aligned}$$

thm-oracles *graph-semantics-preservation-subscript*

maximal-sharing

$$\begin{aligned}
& \text{maximal-sharing } (g :: IRGraph) = \\
& (\forall (n_1 :: nat) \ n_2 :: nat. \\
& \quad n_1 \in \text{true-ids } g \wedge n_2 \in \text{true-ids } g \longrightarrow \\
& \quad (\forall e :: IRExpr. \\
& \quad \quad g \vdash n_1 \simeq e \wedge \\
& \quad \quad g \vdash n_2 \simeq e \wedge \text{stamp } g \ n_1 = \text{stamp } g \ n_2 \longrightarrow \\
& \quad \quad n_1 = n_2))
\end{aligned}$$

tree-to-graph-rewriting

$$\begin{aligned}
& (e_1 :: IRExpr) \sqsupseteq \\
& (e_2 :: IRExpr) \wedge \\
& g_1 :: IRGraph \vdash n :: nat \simeq e_1 \wedge \\
& \text{maximal-sharing } g_1 \wedge \\
& \{n\} \triangleleft g_1 \subseteq (g_2 :: IRGraph) \wedge \\
& g_2 \vdash n \simeq e_2 \wedge \\
& \text{maximal-sharing } g_2 \implies \\
& \text{term-graph-refinement } g_1 \ g_2
\end{aligned}$$

thm-oracles *tree-to-graph-rewriting*

term-graph-refines-term

$$\begin{aligned}
& (g :: IRGraph \vdash n :: nat \trianglelefteq e :: IRExpr) = \\
& (\exists e' :: IRExpr. g \vdash n \simeq e' \wedge e \sqsupseteq e')
\end{aligned}$$

term-graph-evaluation

$$g :: IRGraph \vdash n :: nat \trianglelefteq e :: IRExpr \implies \\ \forall (m :: nat \Rightarrow Value) (p :: Value\ list) v :: Value. \\ [m, p] \vdash e \mapsto v \longrightarrow [g, m, p] \vdash n \mapsto v$$

graph-construction

$$(e_1 :: IRExpr) \sqsupseteq (e_2 :: IRExpr) \wedge \\ (g_1 :: IRGraph) \subseteq (g_2 :: IRGraph) \wedge \\ g_2 \vdash n :: nat \simeq e_2 \implies \\ g_2 \vdash n \trianglelefteq e_1 \wedge term-graph-refinement\ g_1\ g_2$$

thm-oracles *graph-construction*

term-graph-reconstruction

$$g :: IRGraph \oplus e :: IRExpr \rightsquigarrow (g' :: IRGraph, n :: nat) \implies \\ g' \vdash n \simeq e \wedge g \subseteq g'$$

refined-insert

$$(e_1 :: IRExpr) \sqsupseteq (e_2 :: IRExpr) \wedge \\ g_1 :: IRGraph \oplus e_2 \rightsquigarrow (g_2 :: IRGraph, \\ n' :: nat) \implies \\ g_2 \vdash n' \trianglelefteq e_1 \wedge term-graph-refinement\ g_1\ g_2$$

end

theory *SlideSnippets*

imports

Semantics.TreeToGraphThms

Snippets.Snipping

begin

notation (*latex*)

kind ($-\langle\!\langle\!-\!\rangle\!\rangle$)

notation (*latex*)

IRTreeEval.ord-IRExpr-inst.less-eq-IRExpr ($-\longmapsto -$)

abstract-syntax-tree

```
datatype IExpr =  
  UnaryExpr IRUnaryOp IExpr  
| BinaryExpr IRBinaryOp IExpr IExpr  
| ConditionalExpr IExpr IExpr IExpr  
| ParameterExpr nat Stamp  
| LeafExpr nat Stamp  
| ConstantExpr Value  
| ConstantVar (char list)  
| VariableExpr (char list) Stamp
```

tree-semantics

semantics:constant semantics:parameter semantics:unary semantics:binary semantics:leaf

expression-refinement

$$e_1 \sqsubseteq e_2 = (\forall m \ p \ v. [m,p] \vdash e_1 \mapsto v \longrightarrow [m,p] \vdash e_2 \mapsto v)$$

graph2tree

semantics:constant semantics:unary semantics:binary

graph-semantics

$$([g,m,p] \vdash n \mapsto v) = (\exists e. g \vdash n \simeq e \wedge [m,p] \vdash e \mapsto v)$$

graph-refinement

```
graph-refinement g1 g2 =  
(ids g1 ⊆ ids g2 ∧  
(∀ n. n ∈ ids g1 → (∀ e. g1 ⊢ n ≃ e → g2 ⊢ n ≲ e)))
```

translations

$n \leq \text{CONST as-set } n$

graph-semantics-preservation

$$\begin{aligned} & \llbracket e1' \sqsupseteq e2'; \{n'\} \triangleleft g1 \subseteq g2; \\ & g1 \vdash n' \simeq e1'; g2 \vdash n' \simeq e2' \rrbracket \\ & \implies \text{graph-refinement } g1 \ g2 \end{aligned}$$

maximal-sharing

$$\begin{aligned} & \text{maximal-sharing } g = \\ & (\forall n_1 \ n_2. \\ & \quad n_1 \in \text{true-ids } g \wedge n_2 \in \text{true-ids } g \longrightarrow \\ & \quad (\forall e. g \vdash n_1 \simeq e \wedge \\ & \quad \quad g \vdash n_2 \simeq e \wedge \text{stamp } g \ n_1 = \text{stamp } g \ n_2 \longrightarrow \\ & \quad \quad n_1 = n_2)) \end{aligned}$$

tree-to-graph-rewriting

$$\begin{aligned} & e1 \sqsupseteq e2 \wedge \\ & g1 \vdash n \simeq e1 \wedge \\ & \text{maximal-sharing } g1 \wedge \\ & \{n\} \triangleleft g1 \subseteq g2 \wedge \\ & g2 \vdash n \simeq e2 \wedge \text{maximal-sharing } g2 \implies \\ & \text{graph-refinement } g1 \ g2 \end{aligned}$$

graph-represents-expression

$$(g \vdash n \leq e) = (\exists e'. g \vdash n \simeq e' \wedge e \sqsupseteq e')$$

graph-construction

$$\begin{aligned} & e1 \sqsupseteq e2 \wedge g1 \subseteq g2 \wedge g2 \vdash n \simeq e2 \implies \\ & g2 \vdash n \leq e1 \wedge \text{graph-refinement } g1 \ g2 \end{aligned}$$

end