The Name of the Title is Hope

JUNPENG ZHA, Department of Computer Science and Technology, Nanjing University

ACM Reference Format:
Junpeng Zha. 2018. The Name of the Title is Hope. *Proc. ACM Meas. Anal. Comput. Syst.* 37, 4, Article 111 (August 2018), 8 pages. https://doi.org/10.1145/1122445.1122456

In this document, we show how to define transformation rules to prove the correctness of the compiler optimizations.

1 LANGUAGES

The syntax of the language.

$$
\begin{array}{rcl}
\textit{(Addr)}\ \ x \ \in\ \ldots & \textit{(Reg)}\ \ r\ \in\ \ldots & \textit{(Lab)}\ \ l, \mathsf{f}\ \in\ \ldots
\end{array}
$$

$$
\begin{array}{rrcl}
\textit{(Expre)} & e & ::= & r \mid v \mid e + e \mid e - e \mid e * e \\
\textit{(Instr)} & i & ::= & \langle r := e\rangle_l \mid \langle r := [e]\rangle_l \mid \langle [e_1] := e_2\rangle_l \mid \langle \mathtt{skip}\rangle_l \mid \langle \mathtt{print}\ r\rangle_l \\
 & & \mid & \langle \mathtt{jmp}\rangle_l \mid \langle \mathsf{be}\ e\rangle_{l_1, l_2} \\
\textit{(Code)} & C & ::= & \{l \rightsquigarrow i\}^* \quad \text{where } \ l \in \textit{Lab} \\
\textit{(GE)} & ge & ::= & \{x \rightsquigarrow v\}^* \\
\textit{(Init)} & \mathsf{Init} & \in & \textit{GE} \longrightarrow \textit{Mem} \times \textit{Reg}
\end{array}
$$

The program state.

$$
\begin{array}{rrcl}
\textit{(Mem)} & M & ::= & \{x \rightsquigarrow v\}^* \quad \text{where } \ x \in \textit{Addr} \\
\textit{(RegFile)} & R & ::= & \{r \rightsquigarrow v\}^* \quad \text{where } \ r \in \textit{Reg} \\
\textit{(State)} & S & ::= & (M, R)
\end{array}
$$

The semantics of the language.

$$
\frac{C(\mathsf{pc}) = i \qquad i \vdash ((M, R), \mathsf{pc}) \longrightarrow ((M', R'), \mathsf{pc}')}{C \vdash ((M, R), \mathsf{pc}) \longrightarrow ((M', R'), \mathsf{pc}')}
$$

$$
\frac{[\![e]\!]_R = v}{\langle r := e\rangle_l \vdash ((M, R), \mathsf{pc}) \longrightarrow ((M, R\{r \rightsquigarrow v\}), l)}
\qquad
\frac{[\![e]\!]_R = x \qquad M(x) = v}{\langle r := [e]\rangle_l \vdash ((M, R), \mathsf{pc}) \longrightarrow ((M, R\{r \rightsquigarrow v\}), l)}
$$

$$
\frac{[\![e_1]\!]_R = x \qquad [\![e_2]\!]_R = v}{\langle [e_1] := e_2\rangle_l \vdash ((M, R), \mathsf{pc}) \longrightarrow ((M\{x \rightsquigarrow v\}, R), l)}
\qquad
\frac{}{\langle \mathtt{skip}\rangle_l \vdash ((M, R), \mathsf{pc}) \longrightarrow ((M, R), l)}
$$

$$
\frac{[\![e]\!]_R = v \qquad (l = l_1 \ \wedge\ v = 0) \vee (l = l_2 \ \wedge\ v = 1)}{\langle \mathsf{be}\ r\rangle_{l_1, l_2} \vdash ((M, R), \mathsf{pc}) \longrightarrow ((M, R), l)}
\qquad
\frac{R(r) = v}{\langle \mathtt{print}\ r\rangle_l \vdash ((M, R), \mathsf{pc}) \longrightarrow ((M, R), l)}
$$

Author's address: Junpeng Zha, Department of Computer Science and Technology, Nanjing University.

2 FINAL THEOREM

We show the final theorem in this section. Our goal is to prove the correctness of the compiler optimizations.

$$\textit{(Optimizer)} \quad \mathsf{Optimizer} \quad \in \quad \mathit{Code} \times \mathit{GE} \times \mathit{Lab} \rightharpoonup \mathit{Code}$$

A compiler optimization is correct, if the target code generated refines the source code. We define the refinement relation below.

Definition 2.1 (Refinement).

$$\begin{aligned} C' \subseteq_{ge,\mathsf{f}} C \quad ::= \quad & \forall M, R.\ (\mathsf{Init}(ge) = (M, R) \land \mathsf{Safe}(C, (M, R), \mathsf{f})) \\ & \implies (C', (M, R), \mathsf{f}) \subseteq (C, (M, R), \mathsf{f}) \end{aligned}$$

$$\text{where} \quad \mathsf{Safe}(C, (M, R), \mathsf{f}) \ ::= \ \neg(C \vdash (M, R, \mathsf{f}) \rightarrow^* \textbf{abort}).$$

We give the final theorem below.

Theorem 2.2 (Final Theorem: Optimization Ensures Semantics Preserving).

$$\forall C, C', \mathsf{f}, ge.\ \mathsf{Optimizer}(C, ge, \mathsf{f}) = C' \implies C' \subseteq_{ge,\mathsf{f}} C$$

The correctness proof of the Theorem. 2.2 can be divided into the proof of the two lemmas shown below.

Lemma 2.3 (Optimization Correctness).

$$\forall C, ge, \mathsf{f}.\ \mathsf{Optimizer}(C, ge, \mathsf{f}) = C' \implies \exists I, A.\ I \vdash_{ge,\mathsf{f}} (C, A) \sim C'$$

Lemma 2.4 (Soundness of Transformation rule).

$$\forall C, A, ge, \mathsf{f}.\ I \vdash_{ge,\mathsf{f}} (C, A) \sim C' \implies C' \subseteq_{ge,\mathsf{f}} C$$

In Lemma. 2.3, we prove the correctness of the optimization by proving whether the source code, the target code and the analyses result can pass the checking of the transformation rules.

The transformation rule in the form of "$I \vdash_{ge,\mathsf{f}} (C, A) \sim C'$" will be defined in Sec. 4.

The Lemma. 2.4 shows that the soundness of the transformation rule can ensure that the target code refines the source code.

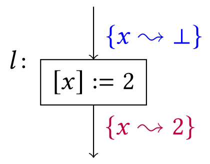
3 THE DEFINITION OF THE ANALYSIS RESULT

We define the result of the analysis *(AnalyRes)* mentioned previously in this section. In this work, we focus on the dataflow analysis mainly.

$$\begin{array}{rrcl} \textit{(AI)} & L & ::= & \ldots \\ \textit{(AIM)} & \mathbb{L} & ::= & \{l_1 \rightsquigarrow L_1, \ldots, l_2 \rightsquigarrow L_n\} \\ \textit{(AnalyRes)} & A & ::= & (\mathbb{L}_i, \mathbb{L}_o) \end{array}$$

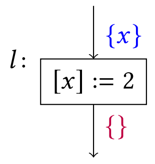
The result of the dataflow analysis gives each node an abstract interpretation. We show the abstract interpretation in the value analysis, the liveness analysis and the common subexpression elimination below.

Value Analysis :



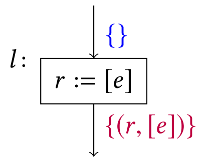
$$(\mathit{AVal}) \quad \hat{v} \;\in\; \mathit{Val} \cup \{\bot, \top\}$$
$$a_v \;\in\; \mathcal{P}((\mathit{Addr} \rightharpoonup \mathit{AVal}) \cup (\mathit{Reg} \rightharpoonup \mathit{AVal}))$$

Liveness Analysis :



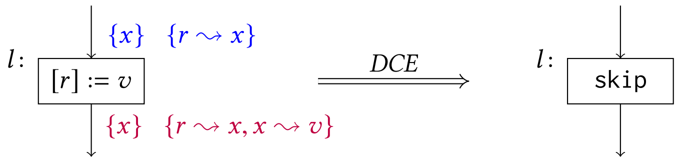
$$a_l \;\in\; \mathcal{P}(\mathit{Addr} \cup \mathit{Reg})$$

Available Expression Analysis :



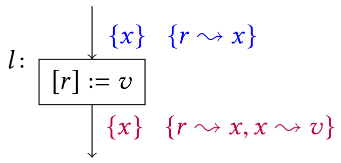
$$(\mathit{VExp}) \quad \mathit{ve} \;::=\; (r, e) \mid (r, [e])$$
$$a_e \;\in\; \mathcal{P}(\mathit{VExp})$$

An analyzer in a compiler optimization may do more than one analysis. For example, the *dead code elimination* optimization in CompCert does both the value analysis and the liveness analysis as shown below.


l:
$[r] := v$
$\{x\}$ $\{r \rightsquigarrow x\}$
$\{x\}$ $\{r \rightsquigarrow x, x \rightsquigarrow v\}$
DCE
l:
skip

The instruction "$[r] := v$" is a dead code, since "$\{r \rightsquigarrow x\}$" says that the register r points to the memory location x, and "$\{x : \circ\}$" says that the memory location x will not be read before the next write.

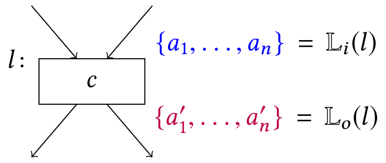
Thus, we define the set of abstract interpretation L *(AISet)*, as the collection of abstract interpretations of each node. We give an example of the specific L below.


l:
$[r] := v$
$\{x\}$ $\{r \rightsquigarrow x\}$
$\{x\}$ $\{r \rightsquigarrow x, x \rightsquigarrow v\}$

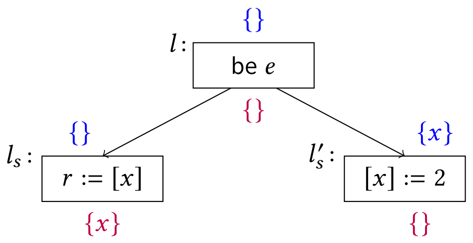
$$L \quad ::= \quad \{\{x\}, \{r \rightsquigarrow x\}\}$$
$$L' \quad ::= \quad \{\{x\}, \{r \rightsquigarrow x, x \rightsquigarrow v\}\}$$

Then, we define the mapping of the set of abstract interpretations $\mathbb{L}$ *(AISetM)*, which is a partial mapping from the label to the set of the abstract interpretations.

Finally, we define the result of the analysis below, which is a pair of *AISetM*. Here, the $\mathbb{L}_i$ maps the label of each node to the set of abstract interpretations, which describe the program state before the execution of the instruction of the node, and the $\mathbb{L}_o$ maps the label of each node to the set of abstract interpretations, which describe the program state after the execution of the instruction of the node. We show the meaning of $\mathbb{L}_i$ and $\mathbb{L}_o$ below.

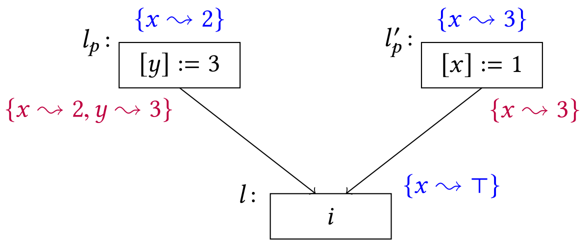

$l:$
c
$\{a_1, \ldots, a_n\} = \mathbb{L}_i(l)$
$\{a'_1, \ldots, a'_n\} = \mathbb{L}_o(l)$

The abstract interpretation, which describes the program state after the execution of the node labelled l, may be different from the abstract interpretation, which describes the program state before the execution of the node l's successor node l_s, as the example shown below.



$$f(\{x\}, r := [x]) = \{\}$$
$$f(\{\}, [x] := 2) = \{x\}$$
$$\{\} \cap \{x\} = \{\}$$

Similarly, The abstract interpretation, which describes the program state before the execution of the node labelled l, may be different from the abstract interpretation, which describes the program state after the execution of the node l's predecessor node l_p, as the example shown below.


$l_p:$
$\{x \rightsquigarrow 2\}$
$[y] := 3$
$\{x \rightsquigarrow 2, y \rightsquigarrow 3\}$
$l'_p:$
$\{x \rightsquigarrow 3\}$
$[x] := 1$
$\{x \rightsquigarrow 3\}$
$l:$
$\{x \rightsquigarrow \top\}$
i

$$f(\{x \rightsquigarrow 2\}, [y] := 3) = \{x \rightsquigarrow 2, y \rightsquigarrow 3\}$$
$$f(\{x \rightsquigarrow 3\}, [x] := 1) = \{x \rightsquigarrow 3\}$$
$$\{x \rightsquigarrow 2, y \rightsquigarrow 3\} \cup \{x \rightsquigarrow 3\} = \{x \rightsquigarrow \top\}$$

4 RELATIONAL TRANSFORMATION RULES

We define the relational transformation rules in this section,

$$
\begin{array}{llll}
(RAst) & p, q & ::= & e =_L e' \mid e =_H e' \mid p[e/r]_H \mid p[e/r]_L \mid e \hookrightarrow_L e' \mid e \hookrightarrow_L e' \\
 & & \mid & p \Rightarrow q \mid p * q \mid p \wedge q \mid p \vee q
\end{array}
$$

Fig. 1. The relational assertion language

$$
\begin{array}{lll}
((M_s, R_s), (M_t, R_t)) \models e =_L e' & ::= & [\![e]\!]_{R_t} = [\![e']\!]_{R_t} \\
((M_s, R_s), (M_t, R_t)) \models e =_H e' & ::= & [\![e]\!]_{R_s} = [\![e']\!]_{R_s} \\
((M_s, R_s), (M_t, R_t)) \models p[e/r]_L & ::= & ((M_s, R_s), (M_t, R_t\{r \rightsquigarrow [\![e]\!]_{R_t}\})) \models p \\
((M_s, R_s), (M_t, R_t)) \models p[e/r]_H & ::= & ((M_s, R_s\{r \rightsquigarrow [\![e]\!]_{R_s}\}), (M_t, R_t)) \models p \\
((M_s, R_s), (M_t, R_t)) \models e \hookrightarrow_L e' & ::= & \exists x.\, [\![e]\!]_{R_t} = x \wedge M_t(x) = [\![e']\!]_{R_t} \\
((M_s, R_s), (M_t, R_t)) \models e \hookrightarrow_H e' & ::= & \exists x.\, [\![e]\!]_{R_s} = x \wedge M_s(x) = [\![e']\!]_{R_s} \\
((M_s, R_s), (M_t, R_t)) \models p \Rightarrow q & ::= & ((M_s, R_s), (M_t, R_t)) \models p \implies ((M_s, R_s), (M_t, R_t)) \models q \\
((M_s, R_s), (M_t, R_t)) \models p * q & ::= & \exists M_1, M_2, M_1', M_2'. \\
 & & \quad \mathrm{dom}(M_1) \cap \mathrm{dom}(M_2) = \emptyset \wedge \mathrm{dom}(M_1') \cap \mathrm{dom}(M_2') = \emptyset \\
 & & \quad \wedge M_1 \cup M_2 = M_s \wedge M_1' \cup M_2' = M_t \\
 & & \quad \wedge ((M_1, R_s), (M_2, R_t)) \models p \wedge ((M_1', R_s), (M_2', R_t)) \models q \\
((M_s, R_s), (M_t, R_t)) \models p \wedge q & ::= & ((M_s, R_s), (M_t, R_t)) \models p \wedge ((M_s, R_s), (M_t, R_t)) \models q \\
((M_s, R_s), (M_t, R_t)) \models p \vee q & ::= & ((M_s, R_s), (M_t, R_t)) \models p \vee ((M_s, R_s), (M_t, R_t)) \models q
\end{array}
$$

Fig. 2. The semantics of the relational assertion language

4.1 Relational Assertion Lanugage

we first define the relational assertion in Fig. 1. And the semantics of the relation assertion language can be found in Fig. 2.

We define an invariant I to describe the relation between the specific analysis result and the relational assertion.

$$(AInv) \quad I \quad \in \quad AI \rightarrow RAsrt$$

Below, we instantiate the invariant I for the *constant propagation*, *dead code elimination* and *common subexpression elimination.*

Constant propagation. We use the a_v as the result of the *value analysis*.

$$
\begin{array}{c}
(AVal) \quad \hat{v} \in Val \cup \{\bot, \top\} \\
a_v \in \mathcal{P}((Addr \rightharpoonup AVal) \cup (Reg \rightharpoonup AVal))
\end{array}
$$

Below, we define the invariant I_c for the correctness proof of the constant propagation pass.

$$
\begin{array}{lll}
I_v(a_v) & ::= & (\forall r \in \mathrm{dom}(a_v).\, (\exists v.\, r =_H v \wedge a_v(r) = v) \vee (r =_H - \wedge a_v(r) \in \{\bot, \top\})) \\
 & & \quad \wedge (\forall x \in \mathrm{dom}(a_v).\, (\exists v.\, x \hookrightarrow_H v \wedge a_v(r) = v) \vee (x \hookrightarrow_H - \wedge a_v(r) \in \{\bot, \top\})) \\
I_c(a_v) & ::= & I_v(a_v) \wedge (\forall r.\, \exists v.\, r =_L v \wedge r =_H v) \wedge (\forall x, v.\, (x \hookrightarrow_H v) \Rightarrow (x \hookrightarrow_L v))
\end{array}
$$

Dead code elimination. We use the a_l as the result of the *liveness analysis*.

$$a_l \in \mathcal{P}(Addr \cup Reg)$$

Below, we define the invariant I_d for the correctness proof of the dead code elimination pass.

$$\begin{array}{rcl} I_d(\{a_v, a_l\}) & ::= & I_v(a_v) \wedge (\forall r \notin a_l.\, \exists v.\, r =_L v \wedge r =_H v) \\ & & \wedge (\forall x, v.\, (x \notin a_l \wedge x \hookrightarrow_H v) \Rightarrow x \hookrightarrow_L v) \end{array}$$

Common subexpression elimination. We use the a_e as the result of the *available expression analysis*.

$$\begin{array}{rl} (VExp) & ve ::= (r, e) \mid (r, [e]) \\ & a_e \in \mathcal{P}(VExp) \end{array}$$

Below, we define the invariant I_e for the correctness proof of the common subexpression elimination.

$$\begin{array}{rcl} I_e(\{a_v, a_e\}) & ::= & I_v(a_v) \wedge (\forall (r, [e]) \in a_e.\, e \hookrightarrow_H r) \wedge (\forall (r, e) \in a_e.\, e =_H r) \\ & & \wedge (\forall r.\, \exists v.\, r =_L v \wedge r =_H v) \wedge (\forall x, v.\, (x \hookrightarrow_H v) \Rightarrow (x \hookrightarrow_L v)) \end{array}$$

4.2 Transformation Rules

We define the transformaiton rules in Fig. 3.

4.3 Soundess of Transformation Rules

Definition 4.1 (Semantics of instruction transformation). $\models \{p\}\ i \sim i'\ \{q\}$ holds iff, for any M_s, R_s, M_t, R_t and pc, if $((M_s, R_s), (M_t, R_t)) \models p$, then

(1) for any M_t', R_t' and pc$'$, if $i' \vdash ((M_t, R_t), \mathsf{pc}) \longrightarrow ((M_t', R_t'), \mathsf{pc}')$, then
- either, there exists M_s' and R_s', such that
$$i \vdash ((M_s, R_s), \mathsf{pc}) \longrightarrow ((M_s', R_s'), \mathsf{pc}') \text{ and } ((M_s', R_s'), (M_t', R_t')) \models q;$$
- or, $i \vdash ((M_s, R_s), \mathsf{pc}) \longrightarrow \textbf{abort}$;

(2) if $i' \vdash ((M_t, R_t), \mathsf{pc}) \longrightarrow \textbf{abort}$, then $i \vdash ((M_s, R_s), \mathsf{pc}) \longrightarrow \textbf{abort}$;

Definition 4.2 (Semantics of code transformation).

$$\begin{array}{rcl} I, A \models C \sim C' & ::= & \forall l \in \mathrm{dom}(C).\ \models \{I(\mathbb{L}_i(l))\}\ C(l) \sim C'(l)\ \{I(\mathbb{L}_o(l))\} \\ & & \text{where } A = (\mathbb{L}_i, \mathbb{L}_o) \end{array}$$

Definition 4.3 (Semantics of well-formed optimization). $I \models_{ge,\mathsf{f}} (C, A) \sim C'$ holds, where $A = (\mathbb{L}_i, \mathbb{L}_o)$, iff

- $(\mathsf{Init}(ge), \mathsf{Init}(ge)) \models I(\mathbb{L}_i(\mathsf{f}))$ holds;
- For any M_s, R_s, M_t, R_t, pc, if $((M_s, R_s), (M_t, R_t)) \models I(\mathbb{L}_i(\mathsf{pc}))$, then the followings hold,
  - for any M_t', R_t' and pc$'$, if $C' \vdash ((M_t, R_t), \mathsf{pc}) \longrightarrow ((M_t', R_t'), \mathsf{pc}')$, then either
  $$C \vdash (M_s, R_s, \mathsf{pc}) \longrightarrow (M_s', R_s', \mathsf{pc}') \text{ and } ((M_s', R_s'), (M_t', R_t')) \models I(\mathbb{L}_i(\mathsf{pc}'));$$
  or, $C \vdash (M_s, R_s, \mathsf{pc}) \longrightarrow \textbf{abort}$;
  - if $C' \vdash ((M_t, R_t), \mathsf{pc}) \longrightarrow \textbf{abort}$, then $C \vdash ((M_s, R_s), \mathsf{pc}) \longrightarrow \textbf{abort}$.

Lemma 4.4 (Soundness of instruction transformation).

$$\forall p, i, i', q.\ \vdash \{p\}\ i \sim i'\ \{q\} \implies \models \{p\}\ i \sim i'\ \{q\}$$

Proof. Prove by discussing each case of "$\vdash \{p\}\ i \sim i'\ \{q\}$". □

$$\frac{\begin{array}{l} I, A \vdash C \sim C' \qquad A = (\mathbb{L}_i, \mathbb{L}_o) \qquad (\mathsf{Init}(ge), \mathsf{Init}(ge)) \models I(\mathbb{L}_i(\mathsf{f})) \\ \text{for all } l \in \mathrm{dom}(C): \\ \qquad I(\mathbb{L}_o(l)) \Rightarrow I(\mathbb{L}_i(l_s)), \text{ for all } \ l_s \in (\mathrm{succ}(C(l))) \end{array}}{I \vdash_{ge,\mathsf{f}} (C, A) \sim C'} \text{(wf-Opt)}$$

$$\frac{I, A \vdash C_1 \sim C_1' \qquad I, A \vdash C_2 \sim C_2'}{I, A \vdash (C_1 \uplus C_2) \sim (C_1' \uplus C_2')} \text{(Link)} \qquad \frac{A = (\mathbb{L}_i, \mathbb{L}_o) \qquad \vdash \{I(\mathbb{L}_i(l))\}\ i \sim i'\ \{I(\mathbb{L}_o(l))\}}{I, A \vdash \{l \rightsquigarrow i\} \sim \{l \rightsquigarrow i'\}} \text{(Ins-Trans)}$$

$$\frac{}{\vdash \{p[e/r]_H[e'/r]_L\}\ \langle r := e\rangle_l \sim \langle r := e'\rangle_l\ \{p\}} \text{(Asgn)} \qquad \frac{}{\vdash \{p[e/r]_H\}\ \langle r := e\rangle_l \sim \langle \mathsf{skip}\rangle_l\ \{p\}} \text{(Asgn-Elim)}$$

$$\frac{\begin{array}{c} (e_1 \hookrightarrow_H -) \wedge p \Rightarrow p_1 \\ p_1 \Rightarrow \exists e_1', e_2'.\, (e_1 \hookrightarrow_H e_1' \wedge e_2 \hookrightarrow_L e_2' \wedge q[e_1'/r]_H[e_2'/r]_L) \end{array}}{\vdash \{p\}\ \langle r := [e_1]\rangle_l \sim \langle r := [e_2]\rangle_l\ \{q\}} \text{(Ld)}$$

$$\frac{\begin{array}{c} (e_1 \hookrightarrow_H -) \wedge p \Rightarrow p_1 \\ p_1 \Rightarrow \exists e_1'.\, (e_1 \hookrightarrow_H e_1' \wedge p'[e_1'/r]_H[e_2/r]_L) \end{array}}{\vdash \{p\}\ \langle r := [e_1]\rangle_l \sim \langle r := e_2\rangle_l\ \{q\}} \text{(Ld-Elim)} \qquad \frac{\begin{array}{c} (e_1 \hookrightarrow_H -) \wedge p \Rightarrow p_1 \\ p_1 \Rightarrow \exists e_1'.\, (e_1 \hookrightarrow_H e_1' \wedge q[e_1'/r]_H) \end{array}}{\vdash \{p\}\ \langle r := [e_1]\rangle_l \sim \langle \mathsf{skip}\rangle_l\ \{q\}} \text{(Ld-Elim2)}$$

$$\frac{\begin{array}{c} (e_1 \hookrightarrow_H -) \wedge p \Rightarrow (e_1 \hookrightarrow_H - * e_2 \hookrightarrow_L - * p_1) \\ e_1 \hookrightarrow_H e_1' * e_2 \hookrightarrow_L e_2' * p_1 \Rightarrow q \end{array}}{\vdash \{p\}\ \langle [e_1] := e_1'\rangle_l \sim \langle [e_2] := e_2'\rangle_l\ \{q\}} \text{(St)}$$

$$\frac{(e_1 \hookrightarrow_H -) \wedge p \Rightarrow (e_1 \hookrightarrow_H - * p) \qquad (e_1 \hookrightarrow_H e_2 * p) \Rightarrow p'}{\vdash \{p\}\ \langle [e_1] := e_2\rangle_l \sim \langle \mathsf{skip}\rangle_l\ \{p'\}} \text{(St-Elim)}$$

Fig. 3. Transformation rules

Lemma 4.5 (Soundness of code transformation).

$$\forall I, A, C, C'.\, I, A \vdash C \sim C' \implies I, A \models C \sim C'$$

Proof. Prove by induction on "$I, A \vdash C \sim C'$".

- Let "$C = (C_1 \uplus C_2)$" and $C' = (C_1' \uplus C_2')$. And we need to prove the following holds.

$$(I, A \models C_1 \sim C_1' \wedge I, A \models C_2 \sim C_2') \implies I, A \models (C_1 \uplus C_2) \sim (C_1' \uplus C_2').$$

  We finish the proof of such case by the definition of the *Semantics of code transformation.*
- Let $C = \{l \rightsquigarrow i\}$, $C' = \{l \rightsquigarrow i'\}$ and $A = (\mathbb{L}_i, \mathbb{L}_o)$. And we need to prove the following holds.

$$\vdash \{I(\mathbb{L}_i(l))\}\ i \sim i'\ \{I(\mathbb{L}_o(l))\} \implies \models \{I(\mathbb{L}_i(l))\}\ i \sim i'\ \{I(\mathbb{L}_o(l))\}$$

  We finish the proof of such case by apply Lemma. 4.4.

□

Lemma 4.6 (Soundness of well-formed optimization).

$$\forall I, A, C, C', ge, \mathsf{f}.\, I \vdash_{ge,\mathsf{f}} (C, A) \sim C' \implies I \models_{ge,\mathsf{f}} (C, A) \sim C'$$

Proof. We prove by unfolding "$I \vdash_{ge,\mathsf{f}} (C, A) \sim C'$". Let "$A = (\mathbb{L}_i, \mathbb{L}_o)$" and we get the followings hold.

$$I, A \vdash C \sim C' \tag{1}$$

$$(\mathsf{Init}(ge), \mathsf{Init}(ge)) \models I(\mathbb{L}_i(\mathsf{f})) \tag{2}$$

$$\begin{aligned}&\text{for all } l \in \mathrm{dom}(C) : \\ &\quad I(\mathbb{L}_o(l)) \Rightarrow I(\mathbb{L}_i(l_s)), \text{ for all } l_s \in \mathrm{dom}(C)\end{aligned} \tag{3}$$

By applying Lemma 4.5 on (3), we have

$$I, A \models C \sim C' \tag{4}$$

We need to prove the followings hold.

- $(\mathsf{Init}(ge), \mathsf{Init}(ge)) \models I(\mathbb{L}_i(\mathsf{f}))$. Prove by applying (1).
- For any M_s, R_s, M_t, R_t, pc, if $((M_s, R_s), (M_t, R_t)) \models I(\mathbb{L}_i(\mathsf{pc}))$, then the followings hold,
  - for any M_t', R_t' and pc', if

    $$C' \vdash ((M_t, R_t), \mathsf{pc}) \longrightarrow ((M_t', R_t'), \mathsf{pc}') \tag{5}$$

    , then either

    $$C \vdash (M_s, R_s, \mathsf{pc}) \longrightarrow (M_s', R_s', \mathsf{pc}') \text{ and } ((M_s', R_s'), (M_t', R_t')) \models I(\mathbb{L}_i(\mathsf{pc}'));$$

    or, $C \vdash (M_s, R_s, \mathsf{pc}) \longrightarrow$ **abort**;
    We finish the proof of such case from (4) and (3).
  - if $C' \vdash ((M_t, R_t), \mathsf{pc}) \longrightarrow$ **abort**, then $C \vdash ((M_s, R_s), \mathsf{pc}) \longrightarrow$ **abort**.
    We finish the proof of such case from (4) and (3).

□