Correctness of compiler optimizations

Ori Lahav       Viktor Vafeiadis

30 August 2017
Compilers do more than mapping source command to machine instructions. In particular, they try to optimize the produced code by performing source-to-source transformations.

Examples of transformations:

- **Read-after-write elimination**
  
  \[
  \begin{align*}
  x &:= 1; \\
  a &:= x;
  \end{align*} \quad \sim \quad \begin{align*}
  x &:= 1; \\
  a &:= 1;
  \end{align*}
  

- **Write-read reordering**
  
  \[
  \begin{align*}
  x &:= 1; \\
  a &:= y;
  \end{align*} \quad \sim \quad \begin{align*}
  a &:= y; \\
  x &:= 1;
  \end{align*}
  
  Such optimizations are sound for sequential programs, but are they sound for concurrent programs? It obviously depends on the concurrency semantics (aka the memory model).
Soundness of compiler optimizations

Definition (Sound transformation)

$P_{src} \rightsquigarrow P_{tgt}$ is **sound** under a memory model $X$ if

$$
\llbracket P_{tgt} \rrbracket_X \subseteq \llbracket P_{src} \rrbracket_X
$$

i.e., if every outcome that is allowed $P_{tgt}$ under $X$ is also an allowed outcome for $P_{src}$ under $X$.

- We will implicitly consider families of transformations (e.g., write-read reordering, read-after-write elimination) that can be applied under any context.
- As before, the compiler is allowed to “lose” behaviors.
transforms under SC

- Reorderings are generally unsound under SC.
- Eliminations of adjacent accesses are sound:

Read-after-write elimination

\[
\begin{align*}
  x &:= 1; \quad \Rightarrow \quad x := 1; \\
  a &:= x; \quad \Rightarrow \quad a := 1;
\end{align*}
\]

Read-after-read elimination

\[
\begin{align*}
  a &:= x; \quad \Rightarrow \quad a := x; \\
  b &:= x; \quad \Rightarrow \quad b := a;
\end{align*}
\]

Write-after-write elimination

\[
\begin{align*}
  x &:= 1; \quad \Rightarrow \quad x := 2; \\
  x &:= 2;
\end{align*}
\]

Write-after-read elimination

\[
\begin{align*}
  a &:= x; \quad \Rightarrow \quad a := x; \\
  x &:= a;
\end{align*}
\]

Soundness of these transformations can be proved:
- via the operational semantics of SC using simulations.
- via the declarative semantics of SC.
Example: read-after-write elimination using the declarative semantics

Place the read immediately after the write in the \texttt{sc} order
Transformations under COH and StrongCOH

- Reorderings of independent adjacent accesses of different locations are sound under COH.

**Write-read reordering**

\[ x := v; \quad \sim \quad a := y; \]
\[ a := y; \quad \sim \quad x := v; \]

**Read-read reordering**

\[ a := x; \quad \sim \quad b := y; \]
\[ b := y; \quad \sim \quad a := x; \]

**Write-write reordering**

\[ x := v; \quad \sim \quad y := v'; \]
\[ y := v'; \quad \sim \quad x := v; \]

**Read-write reordering**

\[ a := x; \quad \sim \quad y := v; \]
\[ y := v; \quad \sim \quad a := x; \]

Soundness of these transformations can be proved:

- via the operational semantics of COH using simulations.
- via the declarative semantics of COH.
Example: read-read reordering using the declarative semantics

\[ a := x; \]
\[ b := y; \]
\[ a := x; \]
\[ b := y; \]

COH: \( po_{loc} \cup rf \cup mo \cup rb \) is acyclic
Reorderings in StrongCOH

StrongCOH : COH $\land \text{po} \cup \text{rf}$ is acyclic

<table>
<thead>
<tr>
<th>Write-read reordering</th>
<th>Read-read reordering</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x := v;$ $\leadsto$ $a := y;$ $\checkmark$</td>
<td>$a := x;$ $\leadsto$ $b := y;$ $\checkmark$</td>
</tr>
<tr>
<td>$a := y;$ $\leadsto$ $x := v;$</td>
<td>$b := y;$ $\leadsto$ $a := x;$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Write-write reordering</th>
<th>Read-write reordering</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x := v;$ $\leadsto$ $y := v';$ $\checkmark$</td>
<td>$a := x;$ $\leadsto$ $y := v;$ $\times$</td>
</tr>
<tr>
<td>$y := v';$ $\leadsto$ $x := v;$</td>
<td>$y := v;$ $\leadsto$ $a := x;$</td>
</tr>
</tbody>
</table>
Example: write-read reordering in RA

Reminder: RA-consistency

$$(po \cup rf)^+|_{loc} \cup mo \cup rb \text{ is acyclic}$$

A useful structure for reordering soundness proofs:

reordering = deordering + sequentialization
Write-read deordering in RA

Reminder: RA-consistency

\[(\text{po} \cup \text{rf})^+|_{\text{loc}} \cup \text{mo} \cup \text{rb} \text{ is acyclic}\]

\[G_{\text{src}} \quad \text{\ldots} \quad w : \bar{W} \times v_x \quad \text{\ldots} \quad G_{\text{tgt}}\]
\[r : R \times y \quad v_y \quad \text{\ldots} \quad w : \bar{W} \times v_x \quad \text{\ldots} \quad r : R \times y \quad v_y\]

Observation: \((G_{\text{src}} \cdot \text{po} \cup G_{\text{src}} \cdot \text{rf})^+ = (G_{\text{tgt}} \cdot \text{po} \cup G_{\text{tgt}} \cdot \text{rf})^+ \cup \{\langle w, r \rangle\}\)
Sequentialization in RA

Reminder: RA-consistency

$$(\text{po} \cup \text{rf})^+|_{\text{loc}} \cup \text{mo} \cup \text{rb} \text{ is acyclic}$$

At the execution graph level, sequentialization adds pairs to po:

$$G_{\text{src} \cdot \text{po}} \subseteq G_{\text{tgt} \cdot \text{po}}$$

This is trivially sound under RA.
(Because increasing po cannot remove cycles.)
Reorderings in RA (exercise)

<table>
<thead>
<tr>
<th>Write-read reordering</th>
<th>Read-read reordering</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x := v; )</td>
<td>( a := x; )</td>
</tr>
<tr>
<td>( a := y; )</td>
<td>( b := y; )</td>
</tr>
<tr>
<td>( \sim )</td>
<td>( \sim )</td>
</tr>
<tr>
<td>( x := v; )</td>
<td>( a := x; )</td>
</tr>
<tr>
<td>( a := y; )</td>
<td>( b := y; )</td>
</tr>
<tr>
<td>( x := v; )</td>
<td>( a := x; )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Write-write reordering</th>
<th>Read-write reordering</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x := v; )</td>
<td>( a := x; )</td>
</tr>
<tr>
<td>( y := v'; )</td>
<td>( y := v; )</td>
</tr>
<tr>
<td>( \sim )</td>
<td>( \sim )</td>
</tr>
<tr>
<td>( y := v'; )</td>
<td>( y := v; )</td>
</tr>
<tr>
<td>( \sim )</td>
<td>( \sim )</td>
</tr>
<tr>
<td>( x := v; )</td>
<td>( a := x; )</td>
</tr>
<tr>
<td>( y := v; )</td>
<td>( y := v; )</td>
</tr>
<tr>
<td>( a := x; )</td>
<td>( a := x; )</td>
</tr>
</tbody>
</table>
Eliminations in RA

Read-after-write elimination

\[ \begin{align*}
x &:= 1; \\
a &:= x;
\end{align*} \quad \leadsto \quad \begin{align*}
x &:= 1; \\
a &:= 1;
\end{align*} \]

Read-after-read elimination

\[ \begin{align*}
a &:= x; \\
b &:= x;
\end{align*} \quad \leadsto \quad \begin{align*}
a &:= x; \\
b &:= a;
\end{align*} \]

Write-after-write elimination

\[ \begin{align*}
x &:= 1; \\
x &:= 2;
\end{align*} \quad \leadsto \quad \begin{align*}
x &:= 2; \\
\end{align*} \]

Write-after-read elimination

\[ \begin{align*}
a &:= x; \\
x &:= a;
\end{align*} \quad \leadsto \quad \begin{align*}
a &:= x;
\end{align*} \]
Write-after-write elimination in RA

Reminder: RA-consistency

\[(\text{po} \cup \text{rf})^+|_{\text{loc}} \cup \text{mo} \cup \text{rb}\] is acyclic

\[\ldots \quad \ldots \quad \ldots \quad \ldots \]

\[w_1 : \overrightarrow{W} \times v_1\]

\[\ldots \quad \ldots \quad \ldots \quad \ldots \]

\[\text{mo}\]

\[w_2 : \overrightarrow{W} \times v_2\]

Place \(w_1\) as the immediate predecessor of \(w_2\) in \(\text{mo}_{\text{src}}\).

Observations:

\[\langle a, w_1 \rangle \in \text{mo}_{\text{src}} \Rightarrow \langle a, w_2 \rangle \in \text{mo}_{\text{tgt}}\]

\[\langle a, w_1 \rangle \in \text{rb}_{\text{src}} \Rightarrow \langle a, w_2 \rangle \in \text{rb}_{\text{tgt}}\]

\[\langle a, w_1 \rangle \in (G_{\text{src}.\text{po}} \cup G_{\text{src}.\text{rf}})^+ \Rightarrow \langle a, w_2 \rangle \in (G_{\text{tgt}.\text{po}} \cup G_{\text{tgt}.\text{rf}})^+\]
Write-after-read elimination in RA

Reminder: RA-consistency

\[(\text{po} \cup \text{rf})^+|_{\text{loc}} \cup \text{mo} \cup \text{rb} \text{ is acyclic}\]

\[\begin{array}{ccc}
\ldots & \ldots & \ldots \\
\text{W} \times v & \text{w} : \text{W} \times v & \text{G}_{\text{src}} \\
r : \text{R} \times v & \text{R} \times v & \text{G}_{\text{tgt}} \\
\ldots & \ldots & \ldots
\end{array}\]
Unsoundness of write-after-read elimination in RA

source:  
\[
\begin{align*}
  x &:= 1; \\
  a &:= \text{FAA}(x, 1); \quad // 1 \\
  b &:= y; \quad // 0
\end{align*}
\]

target:  
\[
\begin{align*}
  x &:= 1; \\
  a &:= \text{FAA}(x, 1); \quad // 1 \\
  b &:= y; \quad // 0
\end{align*}
\]

\[
\begin{align*}
  y &:= 1; \\
  c &:= x; \quad // 1 \\
  x &:= c; \\
  b &:= x; \quad // 2
\end{align*}
\]
### Eliminations in RA

<table>
<thead>
<tr>
<th>Read-after-write elimination</th>
<th>Read-after-read elimination</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x := 1;$ $\sim$ $x := 1;$</td>
<td>$a := x;$ $\sim$ $a := x;$</td>
</tr>
<tr>
<td>$a := x;$ $\sim$ $a := 1;$</td>
<td>$b := x;$ $\sim$ $b := a;$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Write-after-write elimination</th>
<th>Write-after-read elimination</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x := 1;$ $\sim$ $x := 2;$</td>
<td>$a := x;$ $\sim$ $a := x;$</td>
</tr>
<tr>
<td>$x := 2;$ $\sim$</td>
<td>$x := a;$ $\sim$</td>
</tr>
<tr>
<td></td>
<td>$a := x;$ $\times$</td>
</tr>
</tbody>
</table>
Summary:

- We defined soundness of program transformations under a memory model.
- We studied various examples.
- Declarative semantics allows simple arguments for soundness.

Not covered:

- Transformation correctness under catch-fire semantics:
  - We may assume that the source program has no “bad” executions.
  - We have to show that the transformation does not introduce “bad” executions.
Exercise: Sequentialization

- Is sequentialization is sound under the simplified C11 model?
- Is sequentialization is sound under TSO?
Exercise: “Roach-motel” reorderings in C11

Part I – The RA model

- Which reorderings are sound under RA? Consider read-read, read-write, write-read, and write-write reorderings. For each case, either prove soundness or provide a counterexample.
- Are any reorderings involving RMW’s sound? Why?

Part II – The C11 model

Show the soundness of the following transformations under the simplified C11 model.

\[
\begin{align*}
\text{x\_rel} & := v_x; \quad \sim \quad \text{y\_rlx} := v_y; \\
\text{y\_rlx} & := v_y; \quad \sim \quad \text{x\_rel} := v_x \\
\text{a} & := \text{x\_rlx}; \quad \sim \quad \text{b} := \text{y\_acq}; \\
\text{b} & := \text{y\_acq}; \quad \sim \quad \text{a} := \text{x\_rlx};
\end{align*}
\]