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

$$\begin{array}{l} x := 1; \\ a := x; \end{array} \quad \rightsquigarrow \quad \begin{array}{l} x := 1; \\ a := 1; \end{array}$$

Write-read reordering

$$\begin{array}{l} x := 1; \\ a := y; \end{array} \quad \rightsquigarrow \quad \begin{array}{l} a := y; \\ x := 1; \end{array}$$

- 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)

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

$$[\![P_{\mathrm{tgt}}]\!]_{\mathrm{X}} \quad \subseteq \quad [\![P_{\mathrm{src}}]\!]_{\mathrm{X}}$$

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.

Transformations under SC

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

Read-after-write elimination

$$\begin{array}{l} x := 1; \\ a := x; \end{array} \quad \leadsto \quad \begin{array}{l} x := 1; \\ a := 1; \end{array}$$

Read-after-read elimination

$$\begin{array}{l} a := x; \\ b := x; \end{array} \quad \leadsto \quad \begin{array}{l} a := x; \\ b := a; \end{array}$$

Write-after-write elimination

$$\begin{array}{l} x := 1; \\ x := 2; \end{array} \quad \leadsto \quad x := 2;$$

Write-after-read elimination

$$\begin{array}{l} a := x; \\ x := a; \end{array} \quad \leadsto \quad a := x;$$

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

Read-after-write elimination

$$\cdots \left\| \begin{array}{c} \vdots \\ x := v; \\ a := x; \\ \vdots \end{array} \right\| \cdots \quad \rightsquigarrow \quad \cdots \left\| \begin{array}{c} \vdots \\ x := v; \\ a := v; \\ \vdots \end{array} \right\| \cdots$$

$$\begin{array}{ccc} & \vdots & \\ \cdots & \mathtt{W}\,x\ v & \cdots \\ & {\scriptstyle \mathtt{rf}}\big\downarrow & \\ & \mathtt{R}\,x\ v & \\ & \vdots & \end{array} \qquad\qquad \begin{array}{ccc} & \vdots & \\ \cdots & \mathtt{W}\,x\ v & \cdots \\ & & \\ & & \\ & \vdots & \end{array}$$

- Place the read immediately after the write in the `sc` order

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

Write-read reordering

$$\begin{array}{l} x := v; \\ a := y; \end{array} \quad \leadsto \quad \begin{array}{l} a := y; \\ x := v; \end{array}$$

Read-read reordering

$$\begin{array}{l} a := x; \\ b := y; \end{array} \quad \leadsto \quad \begin{array}{l} b := y; \\ a := x; \end{array}$$

Write-write reordering

$$\begin{array}{l} x := v; \\ y := v'; \end{array} \quad \leadsto \quad \begin{array}{l} y := v'; \\ x := v; \end{array}$$

Read-write reordering

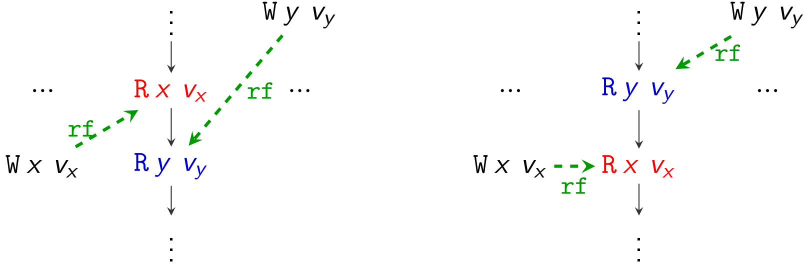
$$\begin{array}{l} a := x; \\ y := v; \end{array} \quad \leadsto \quad \begin{array}{l} y := v; \\ a := x; \end{array}$$

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

$$\ldots \left\| \begin{array}{c}\vdots\\ a := x;\\ b := y;\\ \vdots\end{array} \right\| \ldots \quad \rightsquigarrow \quad \ldots \left\| \begin{array}{c}\vdots\\ b := y;\\ a := x;\\ \vdots\end{array} \right\| \ldots$$



G_{src} $\qquad$ G_{tgt}

COH : $\mathtt{po}|_{\mathtt{loc}} \cup \mathtt{rf} \cup \mathtt{mo} \cup \mathtt{rb}$ is acyclic

Reorderings in StrongCOH

StrongCOH : COH $\wedge$ $\texttt{po} \cup \texttt{rf}$ is acyclic

Write-read reordering

$$\begin{array}{l} x := v; \\ a := y; \end{array} \quad \leadsto \quad \begin{array}{l} a := y; \\ x := v; \end{array} \quad \checkmark$$

Read-read reordering

$$\begin{array}{l} a := x; \\ b := y; \end{array} \quad \leadsto \quad \begin{array}{l} b := y; \\ a := x; \end{array} \quad \checkmark$$

Write-write reordering

$$\begin{array}{l} x := v; \\ y := v'; \end{array} \quad \leadsto \quad \begin{array}{l} y := v'; \\ x := v; \end{array} \quad \checkmark$$

Read-write reordering

$$\begin{array}{l} a := x; \\ y := v; \end{array} \quad \leadsto \quad \begin{array}{l} y := v; \\ a := x; \end{array} \quad \times$$

Example: write-read reordering in RA

Reminder: RA-consistency

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

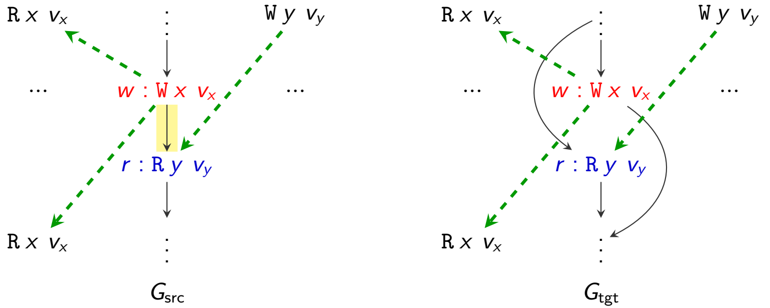
A useful structure for reordering soundness proofs:

reordering $=$ deordering $+$ sequentialization

Write-read deordering in RA

Reminder: RA-consistency

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



Observation: $(G_{\mathsf{src}}.\mathtt{po} \cup G_{\mathsf{src}}.\mathtt{rf})^+ = (G_{\mathsf{tgt}}.\mathtt{po} \cup G_{\mathsf{tgt}}.\mathtt{rf})^+ \cup \{\langle w, r \rangle\}$

Reminder: RA-consistency

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

At the execution graph level, *sequentialization* adds pairs to $\mathtt{po}$:

$$G_{\mathsf{src}}.\mathtt{po} \subseteq G_{\mathsf{tgt}}.\mathtt{po}$$

This is trivially sound under RA.
(Because increasing $\mathtt{po}$ cannot remove cycles.)

Reorderings in RA (exercise)

Write-read reordering

$$\begin{array}{l} x := v; \\ a := y; \end{array} \quad \leadsto \quad \begin{array}{l} a := y; \\ x := v; \end{array} \qquad ?$$

Read-read reordering

$$\begin{array}{l} a := x; \\ b := y; \end{array} \quad \leadsto \quad \begin{array}{l} b := y; \\ a := x; \end{array} \qquad ?$$

Write-write reordering

$$\begin{array}{l} x := v; \\ y := v'; \end{array} \quad \leadsto \quad \begin{array}{l} y := v'; \\ x := v; \end{array} \qquad ?$$

Read-write reordering

$$\begin{array}{l} a := x; \\ y := v; \end{array} \quad \leadsto \quad \begin{array}{l} y := v; \\ a := x; \end{array} \qquad ?$$

Eliminations in RA

Read-after-write elimination

$$\begin{array}{l} x := 1; \\ a := x; \end{array} \quad \rightsquigarrow \quad \begin{array}{l} x := 1; \\ a := 1; \end{array} \quad ?$$

Read-after-read elimination

$$\begin{array}{l} a := x; \\ b := x; \end{array} \quad \rightsquigarrow \quad \begin{array}{l} a := x; \\ b := a; \end{array} \quad ?$$

Write-after-write elimination

$$\begin{array}{l} x := 1; \\ x := 2; \end{array} \quad \rightsquigarrow \quad x := 2; \quad ?$$

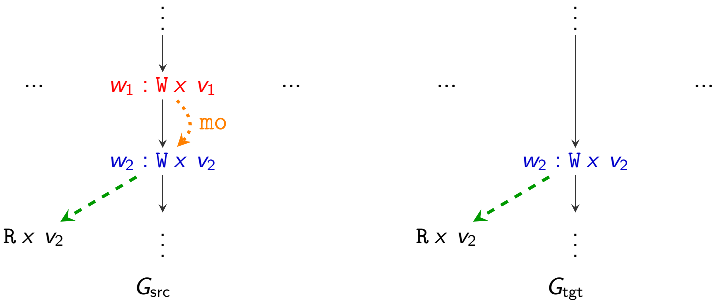
Write-after-read elimination

$$\begin{array}{l} a := x; \\ x := a; \end{array} \quad \rightsquigarrow \quad a := x; \quad ?$$

Write-after-write elimination in RA

Reminder: RA-consistency

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



Place w_1 as the immediate predecessor of w_2 in $\texttt{mo}_{\mathrm{src}}$.

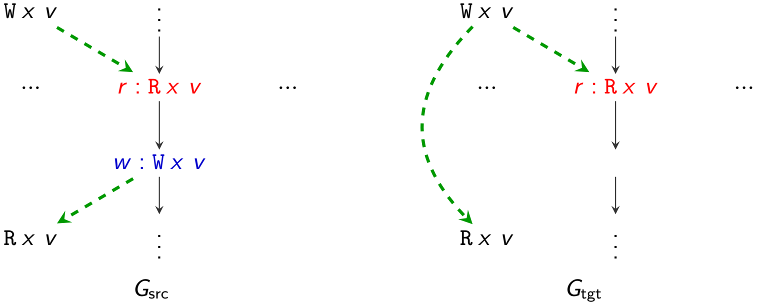
Observations:

$$\langle a, w_1\rangle \in \texttt{mo}_{\mathrm{src}} \Rightarrow \langle a, w_2\rangle \in \texttt{mo}_{\mathrm{tgt}}$$
$$\langle a, w_1\rangle \in \texttt{rb}_{\mathrm{src}} \Rightarrow \langle a, w_2\rangle \in \texttt{rb}_{\mathrm{tgt}}$$
$$\langle a, w_1\rangle \in (G_{\mathrm{src}}.\mathrm{po} \cup G_{\mathrm{src}}.\texttt{rf})^+ \Rightarrow \langle a, w_2\rangle \in (G_{\mathrm{tgt}}.\mathrm{po} \cup G_{\mathrm{tgt}}.\texttt{rf})^+$$

Write-after-read elimination in RA

Reminder: RA-consistency

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



Unsoundness of write-after-read elimination in RA

source:

$$
\begin{array}{l} x := 1; \\ a := \mathbf{FAA}(x, 1); \quad /\!/\, 1 \\ b := y; \quad /\!/\, 0 \end{array}
\;\Bigg\|\;
\begin{array}{l} y := 1; \\ c := x; \quad /\!/\, 1 \\ x := c; \\ b := x; \quad /\!/\, 2 \end{array}
$$

target:

$$
\begin{array}{l} x := 1; \\ a := \mathbf{FAA}(x, 1); \quad /\!/\, 1 \\ b := y; \quad /\!/\, 0 \end{array}
\;\Bigg\|\;
\begin{array}{l} y := 1; \\ c := x; \quad /\!/\, 1 \\ \\ b := x; \quad /\!/\, 2 \end{array}
$$

Eliminations in RA

Read-after-write elimination

$$\begin{array}{l} x := 1; \\ a := x; \end{array} \quad \leadsto \quad \begin{array}{l} x := 1; \\ a := 1; \end{array} \quad ✓$$

Read-after-read elimination

$$\begin{array}{l} a := x; \\ b := x; \end{array} \quad \leadsto \quad \begin{array}{l} a := x; \\ b := a; \end{array} \quad ✓$$

Write-after-write elimination

$$\begin{array}{l} x := 1; \\ x := 2; \end{array} \quad \leadsto \quad x := 2; \quad ✓$$

Write-after-read elimination

$$\begin{array}{l} a := x; \\ x := a; \end{array} \quad \leadsto \quad a := x; \quad ✗$$

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?

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.

$$\begin{array}{l} x_{\mathbf{rel}} := v_x; \\ y_{\mathbf{rlx}} := v_y; \end{array} \quad \leadsto \quad \begin{array}{l} y_{\mathbf{rlx}} := v_y; \\ x_{\mathbf{rel}} := v_x \end{array}$$

$$\begin{array}{l} a := x_{\mathbf{rlx}}; \\ b := y_{\mathbf{acq}}; \end{array} \quad \leadsto \quad \begin{array}{l} b := y_{\mathbf{acq}}; \\ a := x_{\mathbf{rlx}}; \end{array}$$