C/C++ Concurrency and Program Transformations

Proving Reordering correctness

Reordering: $a.b \rightsquigarrow b.a$
Now follow the proof steps:

1. For each consistent execution X' of target program $\mathbb{P}_{\text{tgt}}$, define an execution X of $\mathbb{P}_{\text{src}}$.
2. Show that X is consistent.
3. Show that $O(X) = O(X')$

Step 1.
Given target execution X' we define X as follows:

- $X.\text{E} = X'.\text{E}$
- $X.\text{po} = X'.\text{po} \setminus \{(b, a)\} \cup \{(a, b)\}$
- $X.\text{rf} = X'.\text{rf}$
- $X.\text{mo} = X'.\text{mo}$
- $X.\text{sc} = X'.\text{sc}$

Step 2.
We now show that X is consistent.

We know $X.\mathsf{hb} \subseteq (X'.\mathsf{hb} \setminus \{(b, a)\} \cup \{(a, b)\})$.

(irrHB)
Assume X has an hb cycle due to reordering of a and b.
However, we know a is unique hb-successor of b and b is unique hb-predecessor of a in X'.
Therefore b has no outgoing X.sw edge and a has no incoming X.sw edge.
Therefore $X.\mathsf{hb}$ cycle is not possible involving a and b.
Moreover, $X.\mathsf{hb} \subseteq (X'.\mathsf{hb} \setminus \{(b, a)\} \cup \{(a, b)\})$ ensures there is no other $X.\mathsf{hb}$ cycle.
Therefore $X.\mathsf{hb}$ is irreflexive.

Proving Reordering correctness

Assume X is not coherent.
Assume there exists a $X.\mathsf{rf}; X.\mathsf{hb}$ cycle.
We know $X.\mathsf{hb} \subseteq (X'.\mathsf{hb} \setminus \{(b,a)\} \cup \{(a,b)\})$ and $X.\mathsf{rf} = X'.\mathsf{rf}$.
Therefore there exists a $X'.\mathsf{rf}; (X'.\mathsf{hb} \setminus \{(b,a)\} \cup \{(a,b)\})$ cycle.
It implies a $X'.\mathsf{rf}; X'.\mathsf{hb}$ cycle as $a.\mathsf{Loc} \neq b.\mathsf{Loc}$.
However, we know X' satisfies (CohHBRF).
Hence a contradiction and X satisfies (CohHBRF).

Proving Reordering correctness

Assume X is not coherent.
Assume there exists a $X.\mathsf{rf}; X.\mathsf{hb}$ cycle.
We know $X.\mathsf{hb} \subseteq (X'.\mathsf{hb} \setminus \{(b,a)\} \cup \{(a,b)\})$ and $X.\mathsf{rf} = X'.\mathsf{rf}$.
Therefore there exists a $X'.\mathsf{rf}; (X'.\mathsf{hb} \setminus \{(b,a)\} \cup \{(a,b)\})$ cycle.
It implies a $X'.\mathsf{rf}; X'.\mathsf{hb}$ cycle as $a.\mathsf{Loc} \neq b.\mathsf{Loc}$.
However, we know X' satisfies (CohHBRF).
Hence a contradiction and X satisfies (CohHBRF).

Exercise. Prove the other coherent axioms.

Proving Reordering correctness

(Atomicity)
Assume X does not preserve (Atomicity).
However, from definition, it implies X' violates (Atomicity) which is a contradiction.
Therefore X preserves atomicity.

Proving Reordering correctness

(ConsSC) and (SCreads)
We know $X.\mathsf{sc} = X'.\mathsf{sc}$, and X' preserves (ConsSC) and (SCreads).
Therefore X preserves (ConsSC) and (SCreads).

Allowed Reorderings

Reordering with Release-Acquire accesses

- $\mathsf{WU}_{\sqsupseteq\mathsf{rel}}; \mathsf{WU}_{\sqsubseteq\mathsf{rlx}} \rightsquigarrow \mathsf{WU}_{\sqsubseteq\mathsf{rlx}}; \mathsf{WU}_{\sqsupseteq\mathsf{rel}}$
- $\mathsf{WU}_{\sqsupseteq\mathsf{rel}}; \mathsf{R}_{\sqsubseteq\mathsf{rlx}} \rightsquigarrow \mathsf{R}_{\sqsubseteq\mathsf{rlx}}; \mathsf{WU}_{\sqsupseteq\mathsf{rel}}$

Reordering release or acquire fence with release writes or acquire read

- $\mathsf{F}_{\sqsupseteq\mathsf{rel}}; \mathsf{WU}_{\mathsf{rel}} \rightsquigarrow \mathsf{WU}_{\mathsf{rel}}; \mathsf{F}_{\sqsupseteq\mathsf{rel}}$
- $\mathsf{RU}_{\mathsf{acq}}; \mathsf{F}_{\sqsupseteq\mathsf{acq}} \rightsquigarrow \mathsf{F}_{\sqsupseteq\mathsf{acq}}; \mathsf{RU}_{\mathsf{acq}}$

Reordering fences

- $\mathsf{F}_{\mathsf{rel}}; \mathsf{F}_{\mathsf{acq}} \rightsquigarrow \mathsf{F}_{\mathsf{acq}}; \mathsf{F}_{\mathsf{rel}}$

Reordering Results

$\downarrow a \setminus b \rightarrow$	$R_{\sqsubseteq acq}(\ell)$	$R_{sc}(\ell)$	$W_{na}(\ell)$	$W_{rlx}(\ell)$	$W_{\sqsupseteq rel}(\ell)$	$U_{\sqsubseteq acq}(\ell)$	$U_{\sqsupseteq rel}(\ell)$	F_{acq}	F_{rel}	F_{sc}
$R_{na}(\ell')$	✓	✓	✓	✓	✗	✓	✗	✓	✗	✗
$R_{rlx}(\ell')$	✓	✓	✓	✓	✗	✓	✗	✗	✗	✗
$R_{\sqsupseteq acq}(\ell')$	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗
$W_{\sqsubseteq rel}(\ell')$	✓	✓	✓	✓	✗	✓	✗	✓	✗	✗
$W_{sc}(\ell')$	✓	✗	✓	✓	✗	✓	✗	✓	✗	✗
$U_{\sqsubseteq rel}(\ell')$	✓	✓	✓	✓	✗	✓	✗	✗	✗	✗
$U_{\sqsupseteq acq}(\ell')$	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗
F_{acq}	✗	✗	✗	✗	✗	✗	✗	=	✗	✗
F_{rel}	✓	✓	✓	✗	✓	✗	✓	✓	=	✗
F_{sc}	✗	✗	✗	✗	✗	✗	✗	✗	✗	=

C11 in Compilers

GCC and LLVM compiles C11 concurrency primitives.

Introduce primitives in intermediate representation (IR).

GCC follows C11 concurrency model but LLVM differs.

No optimization on atomic accesses.

LLVM Concurrency Semantics

Concurrency model is different from C11.

Read-write race on non-atomics:

- Concurrent write & read accesses on same location
- One of the accesses is non-atomic.

A program with only read-write races have defined behavior.

- Racy read return **undef**.

Example: Speculative load.

$$\begin{array}{l} \text{if}(f) \\ \quad a = X_{\text{na}}; \end{array} \;\Bigg\|\; X = 1; \quad \rightsquigarrow \quad \begin{array}{l} t = X_{\text{na}}; \\ \text{if}(f) \\ \quad a = t; \end{array} \;\Bigg\|\; X = 1;$$

Source program is non-racy when $f = \mathit{false}$ but the target is read-write racy.

The transformation is disallowed in C11 but allowed in LLVM.

Example: Non-adjacent Read-After-Read.

$$\begin{array}{l} X = 1; \\ Y_{\text{rel}} = 4; \end{array} \left\| \begin{array}{l} a = X_{\text{na}}; \\ \text{if}(Y_{\text{acq}} == 4); \\ \quad c = X_{\text{na}}; \end{array} \right. \rightsquigarrow \begin{array}{l} X = 1; \\ Y_{\text{rel}} = 4; \end{array} \left\| \begin{array}{l} a = X_{\text{na}}; \\ \text{if}(Y_{\text{acq}} == 4); \\ \quad c = a; \end{array} \right.$$

The transformation is allowed in C11 as the source program is already racy.

LLVM disallows the transformation as

- The first read is racy (returns **undef**) but the second read is non-racy (returns constant).
- Read-After-Read elimination replaces a constant value by **undef** i.e. arbitrary value.

Further Reading

Compiler testing and validation papers:

- Compiler testing via a theory of sound optimisations in the C11/C++11 memory model.
PLDI'13.

- Validating Optimizations of Concurrent C/C++ Programs.
CGO'16.