Sound Optimisations in the C11/C++11 Memory Model

November 15, 2012

1 Eliminations of non-atomics

Construction 1.1. From a pre-execution (O', W') of a program P' which is an elimination of a well-defined program P we want to build a candidate execution (O, W). First, we pick an opsem $O \in P$ such that O' is an elimination of O. Then we build a related witness W by taking W', and compute $<_{\mathsf{hb}}$. Finally we modify W as follows:

- RaR: If i is a RaR justified by a read r, then for every write w such that $w <_{\mathsf{rf}} r$, we add $w <_{\mathsf{rf}} i$
- RaW: If i is a RaW justified by a write w, then we add $w <_{\mathsf{rf}} i$
- IR: If i is a IR, then if there is a write w to the same location with $w <_{\mathsf{hb}} i$ we pick an opsem O such that i reads the same value, and add $w <_{\mathsf{rf}} i$ to W.
- OW: No change to W
- WaW: If a is a WaW justified by a write w, then for every read r such that $w <_{\mathsf{rf}} r$ and $a <_{\mathsf{hb}} r$ we replace $w <_{\mathsf{rf}} r$ by $a <_{\mathsf{rf}} r$
- WaR: If a is a WaR, then for every read r of the same value at the same location such that $a <_{\mathsf{hb}} r$ and either $w <_{\mathsf{rf}} r$ with $w <_{\mathsf{hb}} a$ or r reads from no write, we add $a <_{\mathsf{rf}} r$. (replacing $w <_{\mathsf{rf}} r$ if it exists)

Lemma 1.1. *If (O', W') is a pre-execution, then so is (O, W)*

Proof. As we preserve everything but $<_{\mathsf{rf}}$ from W', consistent non-atomic read values is the only predicate that is hard to prove. It is wrong if there is w_1 and w_2 two non-atomic writes at the same location, and i a non-atomic read at the same location with $w_1 <_{\mathsf{hb}} w_2 <_{\mathsf{hb}} i$ and $w_1 <_{\mathsf{rf}} i$. We check every case below to prove that this is impossible.

- RaR:
  If i is not the RaR it is impossible as exactly the same pattern would occur in (O', W'). So we assume that i is a RaR justified by a read r. Because of how the construction works, $w_1 <_{\mathsf{rf}} r$. So because (O', W') is a pre-execution, $w_1 <_{\mathsf{hb}} r$. Three cases must be examined:
  - w_2 and r are not comparable by $<_{\mathsf{hb}}$:
    We use the prefix-closedness of opsemsets to find an opsem $\tilde{O} \in P$ that is a prefix of O that does include w_2 and r but not i. As it does not contain i, the restriction of W to it makes it a pre-execution and it exhibits a data-race between w_2 and r. Absurd as P is well-defined.

- $w_2 <_{\mathsf{hb}} r$:
  Then $w_1 <_{\mathsf{hb}} w_2 <_{\mathsf{hb}} r$ and $w_1 <_{\mathsf{rf}} r$ in (O', W') as well which is impossible as it is a pre-execution.
- $r <_{\mathsf{hb}} w_2$:
  If w_2 is in the same thread as r and i, then $r <_{\mathsf{sb}} w_2 <_{\mathsf{sb}} i$, which is impossible by definition of RaR. Otherwise, there must be a release action a, and an acquire action b such that $r <_{\mathsf{sb}} a <_{\mathsf{hb}} w_2 <_{\mathsf{hb}} b <_{\mathsf{sb}} i$, which is also impossible by definition of RaR

- RaW:
  Again, if the eliminated access is not i, the same problem would happen in (O', W'), so we assume i is the eliminated access. So there must be a write w that justifies i being a RaW and $w <_{\mathsf{rf}} i$. By construction we only add one $<_{\mathsf{rf}}$ edge, so $w = w_1$ and $w_1 <_{\mathsf{sb}} i$. w_2 can't be in the same thread by definition of RaW and if it is in another thread, there must be a release-acquire pair between w_1 and r to justify the $<_{\mathsf{hb}}$ edges, which is again impossible.

- IR:
  Impossible by construction, as we chose a write that is maximal with regards to $<_{\mathsf{hb}}$ as the origin of the $<_{\mathsf{rf}}$ edge.

- OW:
  Because in this case we do not change any $<_{\mathsf{rf}}$ edge, the only hard case (for which the problem is not directly present in (O', W'), is if w_2 is the OW. So there is a write w_3 that justifies it, and $w_2 <_{\mathsf{sb}} w_3$. There are three cases to be looked at:

  - w_3 and i are not comparable by $<_{\mathsf{hb}}$:
    By receptiveness of opsemsets we can pick a new one where i reads the same value as w_2. Then we use prefix-closedness to get a prefix $\tilde{O}$ from that opsem, that contains w_3 and i but nothing after i. By restricting W to that opsem, and replacing $w_1 <_{\mathsf{rf}} i$ by $w_2 <_{\mathsf{rf}} i$, we get a pre-execution. It has a data-race (or an unsequenced-race) between w_3 and i, which is impossible as P is well-defined.
  - $w_3 <_{\mathsf{hb}} i$:
    Then we also have $w_1 <_{\mathsf{hb}} w_3 <_{\mathsf{hb}} i$ and $w_1 <_{\mathsf{rf}} i$ in (O', W') which is impossible as (O', W') is a pre-execution
  - $i <_{\mathsf{hb}} w_3$:
    Either i is in the same thread (impossible by definition of OW) or there is a release-acquire pair between w_2 and w_3 to explain the $<_{\mathsf{hb}}$ edges (impossible for the same reason)

- WaW:
  Either of the writes can be the eliminated one:

  - If w_1 is the WaW:
    It is justified by a write w, with $w <_{\mathsf{hb}} w_2 <_{\mathsf{hb}} i$ and $w <_{\mathsf{rf}} i$ in (O', W'), which is impossible as (O', W') is a pre-execution.
  - If w_2 is the WaW, justified by a write w:
    * If w and w_1 are not comparable by $<_{\mathsf{hb}}$:
      We use the prefix-closedness of opsemsets to find an opsem $\tilde{O} \in P$ that is a prefix of O that does include w and w_1 but not w_2 or i. As it does not contain i, the restriction of W to it makes it a pre-execution and it exhibits a data-race between w and w_1. Absurd as P is well-defined.

* If $w <_{\mathsf{hb}} w_1$:
  Either w_1 is in the same thread as w and w_2 or there is a release-acquire pair between them to account for the $<_{\mathsf{hb}}$ edges. Both are impossible by the definition of WaW
* If $w_1 <_{\mathsf{hb}} w$:
  Then $w_1 <_{\mathsf{hb}} w <_{\mathsf{hb}} i$ and $w_1 <_{\mathsf{rf}} i$ in (O', W') which is impossible because it is a pre-execution.

- WaR:
  - If w_1 is the WaR:
    In (O', W'), $w_2 <_{\mathsf{hb}} i$, so i must be from some write w_0 as (O', W') is a pre-execution. By construction, $w_0 <_{\mathsf{hb}} w_1$ So $w_0 <_{\mathsf{hb}} w_2 <_{\mathsf{hb}} i$ and $w_0 <_{\mathsf{rf}} i$ in (O', W'), which is impossible as it is a pre-execution.
  - If w_2 is the WaR, justified by a read r:
    * If w_1 and r are not comparable by $<_{\mathsf{hb}}$:
      We use the prefix-closedness of opsemsets to find an opsem $\tilde{O} \in P$ that is a prefix of O that does include w_1 and r but not w_2 or i. As it does not contain i, the restriction of W to it makes it a pre-execution and it exhibits a data-race between w_1 and r. Absurd as P is well-defined.
    * If $w_1 <_{\mathsf{hb}} r$:
      By construction, there must exist some write w_3 such that $w_3 <_{\mathsf{rf}} r$. Four cases follow:
      · w_3 and w_1 are not comparable by $<_{\mathsf{hb}}$:
        We use the prefix-closedness of opsemsets to find an opsem $\tilde{O} \in P$ that is a prefix of O that does include w_3 and w_1 but not w_2 or i. As it does not contain i, the restriction of W to it makes it a pre-execution and it exhibits a data-race between w_3 and w_1. Absurd as P is well-defined.
      · $w_3 <_{\mathsf{hb}} w_1$:
        We would have $w_3 <_{\mathsf{hb}} w_1 <_{\mathsf{hb}} r$ and $w_3 <_{\mathsf{rf}} r$ in (O', W'), which is impossible as it is a pre-execution
      · $w_1 <_{\mathsf{hb}} w_3$:
        We would have $w_1 <_{\mathsf{hb}} w_3 <_{\mathsf{hb}} i$ and $w_1 <_{\mathsf{rf}} i$ in (O', W'), which is impossible as it is a pre-execution
      · $w_1 = w_3$:
        As (O', W') is a pre-execution, and $w_1 <_{\mathsf{rf}} r$ in it, w_1 is of the same value as r, and thus of the same value as w_2. So by construction i would read from w_2 instead of w_1.

□

Lemma 1.2. *If (O', W') a pre-execution exhibits an undefined behavior, then so does (O, W) (built by the construction above).*

Proof. We look at the the four possible cases:

- Unsequenced race:
  If a and b are in an unsequenced race in (O', W'), they still are in it in (O, W) as we do not change $<_{\mathsf{sb}}$ in the construction

- Data race:
  The same: $<_{\mathsf{hb}}$ is preserved during the construction

- Indeterminate read:
  We mostly add $<_{\mathsf{rf}}$ edges in the construction to new reads, or we replace already-existing edges. The only exception is in the case for WaR: If w is a WaR justified by a read r, i is an indeterminate read and $w <_{\mathsf{hb}} i$, then $w <_{\mathsf{rf}} i$ in (O, W) and not in (O', W'). However, if there exists w_1 such that $w_1 <_{\mathsf{rf}} i$ in (O', W'), then $w_1 <_{\mathsf{hb}} r$, as (O', W') is a pre-execution. And thus $w_1 <_{\mathsf{hb}} i$ as $r <_{\mathsf{sb}} w <_{\mathsf{hb}} i$, which is impossible as $(O, W)'$ is a pre-execution. So there is no such w_1, and r is also an indeterminate read, in both (O', W') and (O, W)

- Bad mutexes:
  Trivial as we do not affect synchronisation actions in any way.

□

Theorem 1.1. *Let the opsemset P' be an elimination of an opsemset P. If P is well-defined, then so is P' and any execution of P' has the same behavior as some execution of P.*

Proof. First, if P' were to be ill-defined, there would be a pre-execution (O', W') of P' that exhibits an undefined behaviour. By the above lemmata, it is possible to build a corresponding pre-execution (O, W) of P that also exhibits an undefined behaviour, which is impossible as P is assumed to be well-defined. So P' is well-defined
And by the lemmata above, for every execution (O', W') of P' we can build an execution (O, W) of P, that have the exact same observable behavior (by construction, we do not affect in any way synchronisation actions). □

2 Proof of soundness of adding $<_{\mathsf{sb}}$ edges

Theorem 2.1. *Let the opsemset P' be a linearisation of the opsemset P. If P is data-race free then so is P', and any execution of P' has the same observable behaviour as some execution of P*

Proof. Same as structure as for the theorem on eliminations, see below for the appropriate lemmata □

Construction 2.1. We only delete the extra $<_{\mathsf{sb}}$ edges to build O. W is the same as W'.

Lemma 2.1. *If $x <_{\mathsf{hb}} y$ in (O, W), then $x <_{\mathsf{hb}} y$ in (O', W')*

Proof. We keep $<_{\mathsf{sw}}$ and $<_{\mathsf{asw}}$ constant, and the property is true of $<_{\mathsf{sb}}$, so it is also true of $<_{\mathsf{hb}}$. □

Lemma 2.2. *If (O', W') is a pre-exectution then (O, W) is too.*

Proof. We check all clauses of (O, W) being a pre-execution:

- consistent locks:
  $<_{\mathsf{sc}}$ is preserved, and $<_{\mathsf{hb}}$ is more restricted in (O, W) than in (O', W')

- consistent sc-order:
  Same reason as above (plus we preserve $<_{\mathsf{mo}}$).

- consistent mo-order:
  Same reason as above
- Well-formed reads-from mapping:
  $<_{\mathsf{rf}}$ is preserved
- consistent non-atomic read values:
  Only problematic if either
  - $w <_{\mathsf{rf}}^{(O,W)} r$ and not $w <_{\mathsf{hb}}^{(O,W)} r$ while $w <_{\mathsf{hb}}^{(O',W')} r$. In that case w and r are unrelated by $<_{\mathsf{hb}}^{(O,W)}$, so we pick another witness, where r reads from the latest write in $<_{\mathsf{hb}}$ that happens-before it, and we have a contradiction with the fact that P is well-defined
  - Or $w1 <_{\mathsf{hb}}^{(O,W)} w2 <_{\mathsf{hb}}^{(O,W)} r$ and $w1 <_{\mathsf{rf}}^{(O,W)} r$, but then the same problem would have occured in (O', W')
- consistent atomic read values:
  Only problematic if either:
  - $w <_{\mathsf{rf}}^{(O,W)} r$ and $r <_{\mathsf{hb}}^{(O,W)} w$. But in that case, the same problem exist in (O', W')
  - Or $w1 <_{\mathsf{hb}}^{(O,W)} w2 <_{\mathsf{hb}}^{(O,W)} r$ and $w1 <_{\mathsf{rf}} r$, but then the same problem would have occured in (O', W')
- coherent memory use:
  $<_{\mathsf{rf}}$, $<_{\mathsf{mo}}$ are the same in (O, W) and (O', W'), and $<_{\mathsf{hb}}$ is preserved going from (O, W) to (O', W')
- sc-reads restricted:
  Same as above.
- sc-fences heeded:
  Same as above
- RMW-atomicity:
  Same as above

□

Lemma 2.3. *If (O', W') exhibits an undefined behavior, then so does (O, W)*

Proof. We look at the the four possible cases:

- Unsequenced race:
  $<_{\mathsf{sb}}$ edges are only added, not deleted
- Data race:
  $<_{\mathsf{sb}}$ edges (and thus $<_{\mathsf{hb}}$ edges too) are only added, not deleted
- Indeterminate read:
  $<_{\mathsf{rf}}$ is preserved, so obvious
- Bad mutexes:
  $<_{\mathsf{sc}}$ is identical between both pre-execs, and if the pattern with unlock using $<_{\mathsf{sb}}$ is absent in (O', W'), then it is also absent in (O, W)

□

3 Proof of the theorem on introduction of redundant reads

Definition 3.1. A read $a = Rlv$ can be introduced in (O', W') if:

- RaR-introduced:
  - $\forall b$ access to l, $a <_{\mathsf{sb}}^{(O',W')} b$ or $b <_{\mathsf{sb}}^{(O',W')} a$
  - $\exists r = Rlv$
  - $r <_{\mathsf{sb}}^{(O',W')} a$
  - There is no release between r and a w.r.t. $(<_{\mathsf{sb}}^{(O',W')})$
- RaW-introduced
  - $\forall b$ access to l, $a <_{\mathsf{sb}}^{(O',W')} b$ or $b <_{\mathsf{sb}}^{(O',W')} a$
  - $\exists w = Wlv$
  - $w <_{\mathsf{sb}}^{(O',W')} a$
  - There is no release between w and a.

Construction 3.1. To get (O, W) from (O', W'), erase every introduced read, and all relation edges that use them.

Lemma 3.1. *If (O', W') is a pre-execution then (O, W) is a pre-execution*

Proof.
- Consistent non-atomic read-values: this property was true of every load in (O', W') and we preserve $<_{\mathsf{rf}}$ and $<_{\mathsf{hb}}$, so it is still true of every load
- Everything else is trivial too as we touch neither synchronisation actions nor relations between them

□

Lemma 3.2. *If (O', W') a pre-execution has an undefined behavior then (O, W) has one too.*

Proof.
- If (O', W') has a data-race. If that data-race does not involve an introduced read, it is still in (O, W). Otherwise, there is a an introduced read, b the access that justifies it, and w the write that conflicts with it. $b <_{\mathsf{sb}}^{(O',W')} a$, so if $w <_{\mathsf{hb}} b$ we would have $w <_{\mathsf{hb}}^{(O',W')} a$ and no race. So if there is no race in (O, W), $b <_{\mathsf{hb}} w$. Absurd as there is no release between b and a and not $a <_{\mathsf{hb}}^{(O',W')} w$.
- Unsequenced-races involving introduced reads are impossible by definition
- Indeterminate reads not involving introduced reads are clearly propagated to (O, W). If a RaR is an indeterminate read, then the read that justifies it is too, or (O', W') would violate consistent non-atomic read-values. If a RaW is an indeterminate read, then (O', W') violate consistent non-atomic read-values because of the write that justifies it.
- Bad mutexes: we preserve everything related to this.

□

Theorem 3.1. *Let program P' be an introduction of program P.*
Then P well-defined $\implies$ P' well-defined

Proof. Let (O', W') be a pre-execution of P'.
There exists (O, W) a candidate execution of P, and by the above lemmata, (O, W) is also a pre-execution of P.
By the last lemma, (O', W') cannot have undefined behavior or (O, W) would have one. □

Theorem 3.2. *Let P be a program that is well-defined.*
Let P' be a program that is an introduction of P.
Then $\forall (O', W')$ execution of P', $\exists (O, W)$ execution of P, such that $(O, W) \simeq_{\mathsf{vb}} (O', W')$.

Proof. By the previous theorem P' is well-defined.
Let (O', W') be an execution of P'.
By the construction above, we build (O, W) a candidate execution of P with the same observable behavior.
By the lemmata above, (O, W) is a pre-execution of P.
As P is well-defined, (O, W) is actually an execution of it. □

4 Proof of the theorem on reorderings

The proof on reordering works in the same way as the other ones: by building a pre-execution (O, W) of P for every pre-execution (O', W') of P', with the same observable behavior. (and the same undefined behavior if any).

As we only observed reorderings among non-atomics, we only proved those here, and not roach-motel reorderings, where actions can be pushed in critical sections. Roach-motel reordering significantly complicates the proof.

The construction is thus particularly trivial: we just keep the same witness.

Lemma 4.1. *If x and y are actions at the same location, or synchronisation actions then $x <_{\mathsf{hb}}^{(O,W)} y \Leftrightarrow x <_{\mathsf{hb}}^{(O',W')} y$*

Proof. We do not reorder anything with a synchronisation action, and two actions at the same location are not reorderable; and $<_{\mathsf{sw}}$ remains the same as it derives from $<_{\mathsf{rf}}$ and $<_{\mathsf{sc}}$. □

Lemma 4.2. *If (O', W') is a pre-execution, then so is (O, W)*

Proof. $<_{\mathsf{sc}}$, $<_{\mathsf{mo}}$, $<_{\mathsf{rf}}$ are preserved, as well as $<_{\mathsf{hb}}$ on actions to the same location and $<_{\mathsf{hb}}$ on synchronisation actions. □

Lemma 4.3. *If (O', W') exhibits an undefined behaviour, then so does (O, W)*

Proof.
- Unsequenced race: by the definition of reordering
- Data race: $<_{\mathsf{hb}}$ is preserved on actions to the same location
- Indeterminate read: $<_{\mathsf{rf}}$ is preserved
- Bad mutexes: all relations are preserved on synchronisation actions

□