Repairing Sequential Consistency in C/C++11

Ori Lahav
MPI-SWS, Germany *
orilahav@mpi-sws.org

Viktor Vafeiadis
MPI-SWS, Germany *
viktor@mpi-sws.org

Jeehoon Kang
Seoul National University, Korea
jeehoon.kang@sf.snu.ac.kr

Chung-Kil Hur
Seoul National University, Korea
gil.hur@sf.snu.ac.kr

Derek Dreyer
MPI-SWS, Germany *
dreyer@mpi-sws.org

Abstract
The C/C++11 memory model defines the semantics of concurrent memory accesses in C/C++, and in particular supports racy “atomic” accesses at a range of different consistency levels, from very weak consistency (“relaxed”) to strong, sequential consistency (“SC”). Unfortunately, as we observe in this paper, the semantics of SC atomic accesses in C/C++11, as well as in all proposed strengthenings of the semantics, is flawed, in that both suggested compilation schemes to Power are unsound. We propose a better semantics for SC accesses that restores the soundness of the compilation schemes to Power, maintains the DRF-SC guarantee, and provides stronger, more useful, guarantees to SC fences. In addition, we formally prove, for the first time, the correctness of the proposed stronger compilation schemes to Power that preserve load-to-store ordering and avoid “out-of-thin-air” reads.

1. Introduction
The C/C++11 memory model (C11 for short) [7] defines the semantics of concurrent memory accesses in C/C++, of which there are two general types: non-atomic and atomic. Non-atomic accesses are intended for normal data: races on such accesses lead to undefined behavior, thus ensuring that it is sound to subject non-atomic accesses to standard sequential optimizations and reorderings. In contrast, atomic accesses are specifically intended for communication between threads: thus, races on atomics are permitted, but at the cost of imposing restrictions on how such accesses may be merged or reordered during compilation.

The degree to which an atomic access may be reordered with other operations—and more generally, the implementation cost of an atomic access—depends on its consistency level, concerning which C11 offers programmers several options according to their needs. Strongest and most expensive are sequentially consistent (SC) accesses, whose primary purpose is to restore the simple interleaving semantics of sequential consistency [18] if a program (when executed under SC semantics) only has races on SC accesses. This property is called “DRF-SC” and was a main design goal for C11. To ensure DRF-SC, the standard compilation schemes for modern architectures must insert hardware “fence” instructions appropriately into the compiled code, with those for weaker architectures (like Power and ARM) introducing a full (strong) fence adjacent to each SC access.

Weaker than SC atomics are release-acquire accesses, which can be used to perform “message passing” between threads without incurring the implementation cost of a full SC access; and weaker and cheaper still are relaxed accesses, which are compiled down to plain loads and stores at the machine level and which provide only the minimal synchronization guaranteed by the hardware. Finally, the C11 model also supports language-level fence instructions, which provide finer-grained control over where hardware fences are to be placed and serve as a barrier to prevent unwanted compiler optimizations.

In this paper, we are mainly concerned with the semantics of SC atomics (i.e., SC accesses and SC fences), and their interplay with the rest of the model. Since sequential consistency is such a classical, well-understood notion, one might expect that the semantics of SC atomics should be totally straightforward, but sadly, as we shall see, it is not!

The main problem arises in programs that mix SC and non-SC accesses to the same location. Although not common, such mixing is freely permitted by the C11 standard, and has legitimate uses—e.g., as a way of enabling faster (non-SC) reads from an otherwise quite strongly synchronized data structure. Indeed, we know of several examples of code in the wild that mixes SC accesses together with release/acquire or relaxed accesses to the same location: seqLocks [8] and Rust’s crossbeam library [2]. Now consider the following program (see Manerkar et al. [20]):

x :=_sc 1
a :=_acq x_sc #1
b := y_acq #1
c := y_sc #0
d := x_sc #0

(IRIW-acq-sc)
Two compilation schemes have been proposed \cite{6}: the first fixes a problem already noted before the C11 model, which strengthens the semantics of SC coincides with the strengthened model of Batty et al. We prove correctness of compilation to Power. Moreover, it still satisfies the standard DRF-SC convention, whereas an alternative scheme inserts a sync fence after each SC access ("trailing sync" convention). The intent of both schemes is to have a strong barrier between every pair of SC accesses, and thereby enforce sequential consistency on programs containing only SC accesses. Nevertheless, by mixing SC and release-acquire accesses, one can quickly get into trouble, as illustrated by IRIW-acq-sc.

In particular, if one compiles the program into Power using the trailing sync convention, then the behavior is allowed. Since all SC accesses are at the end of the threads, the trailing sync fences have no effect, and the example reduces to IRIW with only acquire reads, which is allowed by the Power memory model. In §2.1, we show further examples illustrating that the other, leading-sync scheme also leads to behaviors in the target of compilation to Power that are not permitted in the source.

Although previous work has found multiple problems in the C11 model (e.g., out-of-thin-air problem \cite{28, 10}, the lack of monotonicity \cite{27}), none of them until now affected the correctness of compilation to the mainstream architectures. In contrast, the IRIW-acq-sc program and our examples in §2.1 show that both the suggested compilation schemes to Power are unsound with respect to the C11 model, thereby contradicting the published results of \cite{6, 24}. Consequently, the strengthened model of Batty et al. \cite{4} is also not efficiently implementable on Power.

In the remainder of the paper, we propose a way to repair the semantics of SC accesses that resolves the problems mentioned above. In particular, our corrected semantics restores the soundness of the suggested compilation schemes to Power. Moreover, it still satisfies the standard DRF-SC theorem: if a program's sequentially consistent executions only ever exhibit races on SC atomic accesses, then its semantics under full C11 is also sequentially consistent. It is worth noting that our correction only affects the semantics of programs mixing SC and non-SC accesses to the same location: we show that, without such mixing, our correction coincides with the strengthened model of Batty et al. \cite{4}.

We also apply two additional, orthogonal, corrections to the C11 model, which strengthen the semantics of SC fences. The first fix corrects a problem already noted before putting all these corrections together, we propose a new model called RC11 (for Repaired C11) that supports nearly all features of the C11 model except consume atomics (§3). We prove correctness of compilation to TSO (§4), Power (§5), and ARMv7 (§6), the soundness of a wide collection of program transformations (§7) and a DRF-SC theorem (§8).

### 2. The Semantics of SC Atomics in C11: What’s Wrong, and How Can We Fix It?

The C11 memory model defines the semantics of a program as a set of consistent executions. Each execution is a graph. Its nodes, E, are called events and represent the individual memory accesses and fences of the program, while its edges represent various relations among these events:

- The sequenced-before (ab) relation, a.k.a. program order, captures the order of events in the program's control flow.
- The modification order (mo) is a union of total orders, one for each memory address, totally ordering the writes to that address. Intuitively, it records for each memory

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure1.png}
\caption{An execution of IRIW-acq-sc yielding the result \( a = b = 1 \land c = d = 0 \).}
\end{figure}
We denote by $R$ the set of all SC events in $E$. The actual definition of $f$ is left for the reader to fill in. We also write $E^{ac}$ for the set of all SC events in $E$.

Given a binary relation $R$, we write $R^3$, $R^+$, and $R^*$ respectively to denote its reflexive, transitive, and reflexive-transitive closures. The inverse relation is denoted by $R^{-1}$. We denote by $R_1; R_2$ the left composition of two relations $R_1, R_2$, and assume that $; \preceq \cup$ and $\setminus$. Finally, we denote by $[A]$ the identity relation on a set $A$. In particular, $[A]; R; [B] = R \cap (A \times B)$.

Based on these three basic relations, let us define some derived relations. First, whenever an acquire or SC read reads from a release or SC write, we say that the write synchronizes with (sw) the read.\(^1\) Next, we say that one event happens before (hb) another event if they are connected by a sequence of sb or sw edges. Formally, hb is taken to be $(sb \cup sw)^+$. For example, in Fig. 1, event $k$ synchronizes with $l$ and therefore $k$ happens-before $l$ and $m$. Lastly, whenever a read event $e$ reads from a write that is sb-before another write $f$, then that event is sb-before $f$. Formally, $rb = (rb^{-1}; mo) \setminus [E]$. (The "\$E\" part is needed so that RMW events do not read before themselves.)

Consistent C11 executions require that hb is irreflexive (i.e., $ab \cup sw$ is acyclic), and further guarantee coherence (aka SC-per-location). Roughly speaking, coherence ensures that (i) the order of writes to the same location according to $mo$ does not contradict hb (COHERENCE-WW); (ii) reads do not read values written in the future (NO-FUTURE-READ and COHERENCE-RW); (iii) reads do not read overwritten values (COHERENCE-WR); and (iv) two hb-related reads from the same location cannot read from two writes in reversed mo-order (COHERENCE-RB). We refer the reader to Prop. 1 in §3 for a precise formal definition of coherence.

Now, to give semantics to SC atomics, C11 stipulates that in consistent executions, there should be a strict total order, $S$, over all SC events, intuitively corresponding to the order in which SC events are executed. This order is required to satisfy a number of conditions (but see Remark 1 below):

(S1) $S$ must include $hb$ restricted to SC events (formally: $[E^{ac}]; hb; [E^{ac}] \subseteq S$);
(S2) $S$ must include $mo$ restricted to SC events (formally: $[E^{ac}]; mo; [E^{ac}] \subseteq S$);
(S3) $S$ must include $rb$ restricted to SC events (formally: $[E^{ac}]; rb; [E^{ac}] \subseteq S$);
(S4-7) $S$ must obey a few more conditions having to do with SC fences.

Remark 1. The S3 condition above, due to Batty et al. [4], is slightly simpler and stronger than the one imposed by the official C11. Crucially, however, all the problems and counterexamples we observe in this section, concerning the C11 semantics of SC atomics, hold for both Batty et al.’s model and the original C11. The reason we use Batty et al.’s version here is that it provides a cleaner starting point for our discussion, and our solution to the problems with C11’s SC semantics will build on it.

Intuitively, the effect of the above conditions is to enforce that, since $S$ corresponds to the order in which SC events are executed, it should agree with the other global orders of events: $hb$, $mo$, and $rb$. However, as we will see shortly, condition (S1) is too strong. Before we get there, let us first look at a few examples to illustrate how the conditions on $S$ interact to enforce sequential consistency.

Consider the classic “store buffering” litmus test:

\[
x := sc 1 \quad y := sc 0 \quad b := x_{sc} \not\in S
\]

Here, the annotated behavior is forbidden by C11. Without loss of generality, assume $x := sc 1$ is before $y := sc 1$ in $S$. By condition S1, since $y := sc 1$ is sequenced before $b := x_{sc}$, it is also before $b := x_{sc}$ in $S$. Thus, by transitivity of $S$, $x := sc 1$ is before $b := x_{sc}$ in $S$. Finally, if the read of $x$ returns 0, then that means it reads-before $x := sc 1$, which by condition S3 means that $S$ must order $b := x_{sc}$ before $x := sc 1$. This would entail a cycle in $S$, which is impossible.

Similarly, C11’s conditions guarantee that the following (variant given in [29] of the) 2+2W litmus test disallows the annotated weak behavior:

\[
x := sc 1 \quad y := sc 2 \quad x := sc 1 \quad y := sc 2 \quad a := rlx \#1 \quad b := x_{rlx} \#1
\]

Because of coherence condition (iii), $mo$—and thus, by S2, also $S$—must order $x := sc 2$ before $x := sc 1$, and $y := sc 2$ before $y := sc 1$; otherwise, the reads at the end of each thread would be reading overwritten values. At the same time, $x := sc 1$ is sequenced before $y := sc 2$, and $y := sc 1$ before $x := sc 2$, and thus by S1, those orderings must be included in $S$ as well. Together, these induce an illegal cycle in $S$.\(^1\)

\(^1\) The actual definition of $sw$ contains further cases, which are not relevant for the current discussion. These are included in our formal model in §3.
strong ordering guarantees on accesses to the same memory provide coherence, hardware memory models provide rather What Went Wrong and How to Fix it

the last thread, is equivalent to an sync beginning of a thread. In the absence of other sync fences, the SC accesses are useless because they are at the sync point, because of the SC accesses are useless because they are at the sync point, non-SC accesses to the same location, every sync fence places in the two accesses. Unfortunately, in the presence of mixed SC and non-SC accesses, the Power compilation schemes do not always ensure that a sync exists between hb-related SC accesses. Specifically, if we follow the trailing sync convention, the hb-path (in Fig. 1) from k to m starting with an wb edge avoids the sync fence placed after k. Conversely, if we follow the leading sync convention, the hb-path (in Fig. 2) from k to m ending with an wb edge avoids the fence placed before m. The result is that S1 enforces more ordering than the hardware provides!

So, if requiring that hb (on SC events) be included in S is too strong a condition, what should we require instead? The essential insight is that, according to either compilation scheme, we know that a fence will necessarily exist between SC accesses a and b if the hb path from a to b starts and ends with an sb edge. Secondarily, note that if a is a write and b is a read that reads directly from a, then the hardware will preserve the ordering anyway. These two observations lead us to replace condition S1 with the following:

(S1fix) S must include hb, restricted to SC events, and further restricted to hb-paths that either start and end with a sb edge, or consist of a single rf edge (formally: \([E^{sc}] \subseteq \{rb \cup sb; hb; sb \cup rb; \} \subseteq S\).

We note that condition S1fix, although weaker than S1, suffices to rule out the weak behaviors of the basic litmus tests (i.e., SB and 2+2W). In fact, just to rule out these behaviors, it suffices to require sb (on SC events) to be included in S.

Figure 2. A C11 execution of Z6.U. The initialization of y is omitted as it is not relevant.

Let us now move to the IRIW-acq-sc program from the introduction, whose annotated behavior is also forbidden by C11. To see that, suppose without loss of generality that S(p, k) in Fig. 1. We also know that S(k, m) because of happens-before via l (S1). Thus, by transitivity, S(p, m).

However, if the second thread reads y = 0, then m reads-before p, in which case S(m, p) (S3), and S has a cycle.

2.1 First Problem: Compilation to Power is Broken

As we saw in the introduction, the IRIW-acq-sc example demonstrates that the trailing sync compilation is unsound for the C11 model. We will now see an example showing that the leading sync compilation is also unsound. Consider the following behavior, where all variables are zero-initialized and FAI(y) represents an atomic fetch-and-increment of y returning its value before the increment:

\[
x := \text{sc} \ 1
y := \text{rel} \ 1
z := \text{sc} \ 3
a := x_{\text{sc}} \ #0
\]

(Z6.U)

We will show that the behavior is disallowed according to C11, but allowed by its compilation to Power.

Fig. 2 depicts the only execution yielding the behavior in question. The rf and mo edges are forced because of coherence: even if all accesses in the program were relaxed-atomic, they would have to go this way. S(k, m) holds because of condition S1 (k happens-before l, which happens-before m); S(m, o) holds because of condition S2 (m precedes o in modification order); S(o, p) holds because of condition S1 (o happens-before p). Finally, since p reads x = 0, we have that p reads-before k, so by S3, S(p, k), thus forming a cycle in S.

Under the leading sync compilation to Power, however, the behavior is allowed (albeit not yet observed on existing implementations). Intuitively, all but one of the sync fences because of the SC accesses are useless because they are at the beginning of a thread. In the absence of other sync fences, the only remaining sync fence, due to the \(a := x_{\text{sc}}\) load in the last thread, is equivalent to an \(w_{\text{sc}}\) fence (cf. [15, §7]).

A similar example can be constructed without SC RMW instructions, using SC fences instead (see Appendix A.1).

What Went Wrong and How to Fix it

Generally, in order to provide coherence, hardware memory models provide rather strong ordering guarantees on accesses to the same memory location. Consequently, for conditions S2 and S3, which only enforce orderings between accesses to the same location, ensuring that compilation preserves these conditions is not difficult, even for weaker architectures like Power and ARM.

When, however, it comes to ensuring a strong ordering between accesses of different memory locations, as S1 does, compiling to weaker hardware requires the insertion of appropriate memory fence instructions. In particular, for Power, to enforce a strong ordering between two hb-related accesses, there should be a Power sync fence occurring somewhere in the hb-path (the sequence of sb and wb edges) connecting the two accesses. Unfortunately, in the presence of mixed SC and non-SC accesses, the Power compilation schemes do not always ensure that a sync exists between hb-related SC accesses. Specifically, if we follow the trailing sync convention, the hb-path in Fig. 1 starting with an wb edge avoids the sync fence placed after k. Conversely, if we follow the leading sync convention, the hb-path in Fig. 2 from k to m ending with an wb edge avoids the fence placed before m. The result is that S1 enforces more ordering than the hardware provides!

So, if requiring that hb (on SC events) be included in S is too strong a condition, what should we require instead? The essential insight is that, according to either compilation scheme, we know that a fence will necessarily exist between SC accesses a and b if the hb path from a to b starts and ends with an sb edge. Secondarily, note that if a is a write and b is a read that reads directly from a, then the hardware will preserve the ordering anyway. These two observations lead us to replace condition S1 with the following:

(S1fix) S must include hb, restricted to SC events, and further restricted to hb-paths that either start and end with a sb edge, or consist of a single rf edge (formally: \([E^{sc}] \subseteq \{rb \cup sb; hb; sb \cup rb; \} \subseteq S\).

We note that condition S1fix, although weaker than S1, suffices to rule out the weak behaviors of the basic litmus tests (i.e., SB and 2+2W). In fact, just to rule out these behaviors, it suffices to require sb (on SC events) to be included in S.

Further, we note that in the absence of mixing of SC and non-SC accesses to the same location, every hb-path between two SC accesses that does not go through another SC access is either a direct rf-edge or has to start and end with an sb edge, in which case the two conditions coincide.

Fixing the Model

Before formalizing our fix, let us first rephrase conditions S1–S3 in the more concise style suggested by Batty et al. [4]. Instead of expressing them as separate conditions on a total order S, Batty et al. require a single acyclicity condition, namely that \([E^{sc}] \subseteq \{rb \cup sb; hb; sb \cup rb; \} \subseteq S\) be acyclic. (In general, acyclicity of \(\bigcup R_i\) is equivalent to the existence of a total order \(S\) that contains \(R_1, R_2, \ldots\))

We propose to correct the condition by replacing hb with sb \(\cup sb; hb; sb \cup rf\), thus requiring the relation

\[
psc_1 \triangleq [E^{sc}] \subseteq \{sb \cup sb; hb; sb \cup rf \cup mo \cup rb\}; [E^{sc}]
\]
to be acyclic instead. Our weaker condition suffices to prove the most basic usefulness criterion for SC accesses, namely the DRF-SC theorem. It turns out that even the acyclicity of \([E^{sc}] \cup (sb \cup rf \cup mo \cup rb)\); \([E^{sc}]\) suffices to prove DRF-SC. This should not be very surprising: in the extreme case when all accesses are SC, this corresponds exactly to the declarative definition of sequential consistency [25].

2.2 Second Problem: SC Fences are Too Weak

We move on to a second problem, this time involving SC fences. Denote by \(F^{sc}\) the set of SC fences in \(E\). The condition of Batty et al. [5] for the full model is that

\[
psc_{Batty} \triangleq ([E^{sc}] \cup [F^{sc}]_{sb}) : (hb \cup mo \cup rb) : ([E^{sc}] \cup [F^{sc}]_{sb})
\]

is acyclic. This condition generalizes the earlier condition by forbidding \((hb \cup mo \cup rb)-cycles even between non-SC accesses provided they are preceded/followed by an SC fence. This condition rules out weak behaviors of examples such as SB and 2+2W where all accesses are relaxed and SC fences are placed between them in all threads.

In general, one might expect that inserting an SC fence between every two instructions restores sequential consistency. This holds for hardware memory models, such as TSO, Power, and ARM, for programs with aligned word-sized accesses (for their analogue of SC fences), but does not hold neither in the original C11 model nor in its strengthening [5] for two reasons. The first reason is that C11 declares that programs with racy non-atomic accesses have undefined behavior, and even if fences are placed everywhere such races may exist. There is, however, another way in which putting fences everywhere in C11 does not restore sequential consistency, even if all the accesses are atomic. Consider the following program:

\[
x := rlx_1 \quad \quad \begin{cases} \quad a := rlx_1 \quad \quad b := y_{rlx_1} \quad \quad c := x_{rlx_1} \quad \quad \text{(RWC+syncs)} \\ \quad \quad \text{fence}_{sc} \quad \quad \quad \text{fence}_{sc} \quad \quad \quad \text{fence}_{sc} \end{cases}
\]

The annotated behavior is allowed according to the model of Batty et al. [5]. Fig. 3 depicts a consistent execution yielding this behavior, as the only \(psc_{Batty}\)-edge is from \(f_1\) to \(f_2\). Yet, this behavior is disallowed by all implementations of C11. We believe that this is a serious omission of the standard rendering the SC fences too weak, as they cannot be used to enforce sequential consistency. This weakness has also been observed in an C11 implementation of the Chase-Lev deque by Lê et al. [19], who report that the weak semantics of SC fences in C11 requires them to unnecessarily strengthen the access modes of certain relaxed writes to SC. (In the context of the RWC+syncs, it would amount to making the write to \(x\) in the first thread into an SC write.)

Remark 2 (Itanium). A possible justification for this weakness of the standard is that there was a fear that the implementation of fences on Itanium does not guarantee the property of restoring sequential consistency when fences are inserted everywhere. This fear is unfounded for two independent reasons. First, all atomic accesses are compiled to release/acquire Itanium accesses on which Itanium fences guarantee ordering. Second, even if this were not the case, Itanium implementations provide multi-copy atomicity, and thus cannot yield the weak outcome of IRIW even without fences [13, §3.3.7.1]. Nevertheless, the whole discussion is probably not very relevant any more, as Itanium is becoming obsolete.

Fixing the Semantics of SC Fences

Analyzing the execution of RWC+syncs, we note that there is a \(sb; rb; rf\) path from \(f_2\) to \(f_1\), but this path does not contribute to \(psc_{Batty}\). Although both \(rb\) and \(rf\) edges contribute to \(psc\), their composition \(rb; rf\) does not.

To repair the model, we define the extended coherence order, \(eco\), to include the read-from relation, \(rf\), the modification order, \(mo\), the reads-before relation, \(rb\), and also all the compositions of these relations with one another—namely, all orders forced because of the coherence axioms. Our condition then becomes that

\[
psc_2 \triangleq ([E^{sc}] \cup [F^{sc}]_{sb}) : (sb \cup sb; rb; sb \cup eco) : ([E^{sc}] \cup [F^{sc}]_{sb; F^{sc}})
\]

is acyclic, where \(eco \triangleq (rf \cup mo \cup rb)^+\). This stronger condition rules out the weak behavior of RWC+syncs because there are \(sb; rb; rf\) paths from one fence to another and vice versa (in one direction via the \(x\) accesses and in the other direction via the \(y\) accesses).

Intuitively speaking, compilation to Power remains correct with this stronger model, since \(eco\) exists only between accesses to the same location, on which Power provides strong ordering guarantees.

Now it is easy to see that, given a program without non-atomic accesses, placing an SC fence between every two accesses guarantees SC. By the definition of SC, it suffices to show that \(eco \cup sb\) is acyclic. Consider a \(eco \cup sb\) cycle. Since \(eco\) and \(sb\) are irreflexive and transitive, the cycle necessarily has the form \((eco; sb)^+\). Between every two \(eco\) steps, there must be an SC fence. So in effect, we have a cycle in \((eco; sb; [F^{sc}]; sb)^+\), which can be regrouped to a cycle in \(([F^{sc}]; sb; eco; sb; [F^{sc}])^+\), which is forbidden by our model.
behavior is forbidden, but it will become allowed after eliminating \( x := \text{sc} \).

\[
\begin{align*}
  a & := x_{\text{acq}} / 2 & x & := \text{sc} 1 \\
  b & := y_{\text{acq}} / 0 & x & := \text{sc} 2 \\
  c & := x_{\text{sc}} / 0 & y & := x_{\text{sc}} 1 \\
\end{align*}
\]  
(WWmerge)

Similarly, the unsoundness of eliminating a repeated SC read is witnessed by the following program. Again, the annotated behavior is forbidden, but it will become allowed after replacing \( b := x_{\text{sc}} \) by \( b := a \).

\[
\begin{align*}
  a & := x_{\text{sc}} / 1 & a & := x_{\text{sc}} / 1 \\
  b & := x_{\text{sc}} / 1 & b & := x_{\text{sc}} / 1 \\
  c & := x_{\text{sc}} / 0 & c & := x_{\text{sc}} / 0 \\
  d & := y_{\text{acq}} / 0 & d & := y_{\text{acq}} / 0 \\
\end{align*}
\]  
(RRmerge)

The problem here is that these transformations remove an \( \text{sb} \)-edge, and thus removes an \( \text{sb}: \text{hb} \); \( \text{sb} \) path between two SC accesses.\(^2\) Note that the removed \( \text{sb} \)-edges are all edges between same location accesses. Thus, supporting these transformations can be achieved by a slight weakening of our model. The solution is to replace \( \text{sb}: \text{hb} \); \( \text{sb} \) in \( \text{psc} \) by \( \text{sb}\not\in\text{loc}: \text{hb} ; \text{sb}\not\in\text{loc} \), where \( \text{sb}\not\in\text{loc} \) denotes \( \text{sb} \)-edges between accesses to different locations. Our final condition becomes acyclicity of the following relation:

\[
\begin{align*}
  \text{psc} & \triangleq \left( [\text{ec}] \cup [\text{psc}] ; \text{hb} \right) ; (\text{sb} \cup \text{ec} \cup \text{sb}\not\in\text{loc} ; \text{hb} ; \text{sb}\not\in\text{loc} ) ; ([\text{ec}] \cup \text{hb} ; [\text{psc}]) \\
\end{align*}
\]

We note that this change does not affect programs that do not mix SC and non-SC accesses to the same location.

### 2.4 A Final Problem: Out-of-Thin-Air Reads

The C11 memory model suffers from a major problem, known as the “out-of-thin-air problem” [28, 10]. Designed to allow efficient compilation and many optimization opportunities for relaxed accesses, the model happened to be too weak, admitting “thin-air” behaviors, which no implementation exhibits. The standard example is load buffering with some form of dependencies:

\[
\begin{align*}
  a & := x_{\text{rlx}} / 1 & b & := y_{\text{rlx}} / 1 \\
  \text{if} (a) y & := x_{\text{rlx}} a & \text{if} (b) x & := x_{\text{rlx}} b \\
\end{align*}
\]  
(LB+deps)

In this program, the formalized C11 model by Batty et al. [7] allows reading \( a = b = 1 \) even though the value does not appear in the program. The reason is that the execution where both threads read and write the value 1 is consistent: each read reads from the write of the other thread. As one might expect, such behaviors are very problematic because they invalidate almost all forms of formal reasoning about the programs. In particular, the example above demonstrates a violation of DRF-SC, the most basic guarantee that users of C11 were intended to assume: LB+deps has no races under sequential consistency, and yet has some non-SC behavior.

Fixing the model in a way that forbids all out-of-thin-air behaviors and still allows the most efficient compilation is

\(^2\)To assist the reader, execution graphs are depicted in Appendix A.2.
3. The Proposed Memory Model

In this section, we formally define our proposed corrected version of the C11 model, which we call RC11. Similar to C11, the RC11 model is given in a “declarative” style in three steps: we associate a set of graphs (called executions) to every program (§3.1), filter this set by imposing a consistency predicate (§3.2), and finally define the outcomes of a program based on the set of its consistent executions (§3.3). At the end of the section, we compare our model with C11 (§3.4).

Before we start, we introduce some further notation. Given a binary relation $R$, $\text{dom}(R)$ and $\text{codom}(R)$ denote its domain and codomain. Given a function $f$, $\text{dom}(f)$ denotes the set of $f$-equivalent pairs ($\text{dom}(f) = \{ (a, b) \mid f(a) = f(b) \}$), and $\text{codom}(f)$ denotes the restriction of $R$ to $f$-equivalent pairs ($\text{codom}(f) = \{ (a, b) \mid f(a) = f(b) \}$). When $R$ is a strict partial order, $\text{dom}(R)$ denotes the set of all immediate $R$-edges, i.e., pairs $(a, b) \in R$ such that for every $c$, $(c, b) \in R$ implies $(c, a) \in R$, and $(a, c) \in R$ implies $(b, c) \in R$.

We assume finite sets Loc and Val of locations and values. We use $x, y, z$ as metavariables for locations and $v$ for values.

The model supports several modes for accesses and fences, partially ordered by $\sqsubset$ as follows:

$$\text{na} \rightarrow \text{rlx} \rightarrow \text{rel} \rightarrow \text{acqrel} \rightarrow \text{sc}$$

3.1 From Programs to Executions

First, the program is translated into a wide set of executions. An execution $G$ consists of:

1. a finite set of events $E \subseteq \mathbb{N}$ containing a distinguished set $E_0 = \{ a_0 \mid x \in \text{Loc} \}$ of initialization events. We use $a, b, \ldots$ as metavariables for events.

2. a function $\text{lab}$ assigning a label to every event in $E$. Labels are of one of the following forms:
   - $R^a(x, v)$ where $a \in \{ \text{na, rlx, acq, sc} \}$.
   - $W^a(x, v)$ where $a \in \{ \text{na, rlx, rel, sc} \}$.
   - $F^a$ where $a \in \{ \text{acq, rel, acqrel, sc} \}$.

We assume that $\text{lab}(a) = W(a, x, 0)$ for every $a \in E_0$. $\text{lab}$ naturally induces functions $\text{typ}, \text{mod}, \text{loc}, \text{val}$, and $\text{val}_x$ that return (when applicable) the type ($R, W$ or $F$), mode, location, and read/written value of an event.

For $T \in \{ R, W, F \}$, $T$ denotes the set $\{ e \in E \mid \text{typ}(e) = T \}$. We also concatenate the event sets notations, use subscripts to denote the accessed location, and superscripts for access modes (e.g., $W[\text{loc}(a)]$ and $W[\text{loc}(a)]^{\text{rel}}$). This results in $\text{val}_x(a)$ and $\text{val}_x(a)$ for every $a \in W$ with $\text{loc}(a) = x$, and $\text{mod}(a) \sqsupseteq \text{rel}$.

3. a strict partial order $\text{sb} \subseteq E \times E$, called $\text{sequenced-before}$, which orders the initialization events before all other events, i.e., $E_0 \subseteq \text{sb}$.

4. a binary relation $\text{rmw} \subseteq \{ \text{wb} \}$, called read-modify-write-pairs, such that for every $(a, b) \in \text{rmw}$, $(\text{mod}(a), \text{mod}(b))$ is one of the following:
   - $(\text{rlx, rlx})$ ($\text{RMW}^{\text{rlx}}$)
   - $(\text{acq, rel})$ ($\text{RMW}^{\text{acqrel}}$)
   - $(\text{acq, rlx})$ ($\text{RMW}^{\text{acq}}$)
   - $(\text{sc, sc})$ ($\text{RMW}^{\text{sc}}$)

We denote by $A_\times$ the set of all events in $E$ that are a part of an rmw edge (that is, $A_\times = \text{dom}(\text{rmw}) \cup \text{codom}(\text{rmw})$).

5. a binary relation $\text{rf} \subseteq \{ \text{wb} \} = \{ \text{lab} \} : \{ R \}$ called reads-from, satisfying (i) $\text{val}_x(a) = \text{val}_x(b)$ for every $(a, b) \in \text{rf}$; and (ii) $a_1 = a_2$ whenever $(a_1, b), (a_2, b) \in \text{rf}$.

6. a strict partial order $\text{mo}$ on $\text{wb}$, called modification order, which is a disjoint union of relations $\{ \text{mo}_x \}_{x \in \text{Loc}}$, such that each $\text{mo}_x$ is a strict total order on $\text{wb}_x$.

In what follows, to resolve ambiguities, we may include a prefix “$G$,” to refer to the components of an execution $G$.

Executions of a given program represent prefixes of traces of shared memory accesses and fences that are generated by the program. In this paper, we only consider “partitioned” programs of the form $\| e \in T_{\text{id}} c_i$, where $T_{\text{id}}$ is a finite set of thread identifiers, $\|$ denotes parallel composition, and...
each $c_i$ is a sequential program. Then, the set of executions associated with a given program is defined by induction over the structure of sequential programs. We do not define formally this construction (it depends on the particular syntax and features of the source programming language). In this initial stage the read values are not restricted whatsoever (and $rf$ and $mo$ are arbitrary). Note that the set of executions of a program $P$ is taken to be prefix-closed: an $sb$-prefix of an execution of $P$ (which includes at least the initialization events) is also considered to be an execution of $P$. By full executions of $P$, we refer to executions of $P$ that represent traces generated by the whole program $P$.

We show an example of an execution in Fig. 5. This is a full execution of the $Z6.U$ program, and is essentially the same as the C11 execution shown in Fig. 2, except that for convenience our executions represent RMWs differently from C11 executions. Here each RMW is represented as two events, a read and a write, related by the $rmw$ relation, whereas in C11 they are represented by single $RMW$ events, which act as both the read and the write of the RMW. Our choice is in line with the POWER and ARM memory models, and simplifies a bit the formal development (e.g., the definition of receptiveness).

### 3.2 Consistent Executions

The main part of the memory model is filtering the consistent executions among all executions of the program. The first obvious restriction is that every read should read some written value (formally, $R \subseteq \text{codom}(rf)$). We refer to such executions as complete. Next, the model defines several constraints with the help of a number of derived relations.

- $rs \triangleq [w]; sb|_{loc}; [w|_{\text{local}}]; (rf; rmw)^*$ (release sequence)
- $sw \triangleq [E|_{\text{local}}]; (F; sb); rs; rf; [R|_{\text{local}}]; (sb; [F]|_{\text{local}}); [E|_{\text{atomic}}]$ (synchronizes with)
- $hb \triangleq (sb \cup sw)^+$ (happens-before)

An important derived relation is the happens-before order ($hb$), which intuitively records when an event is globally perceived as occurring before another one. Happens-before is defined in terms of two more basic definitions. First, following [27], the release sequence ($rs$) of a write contains the write itself and all later writes to the same location in the same thread, as well as all RMWs that recursively read from such writes. Next, a release event $a$ synchronizes with ($sw$) an acquire event $b$, whenever $b$ (or, in case $b$ is a fence, some $sb$-prior read) reads from the release sequence of $a$ (or in case $a$ is a fence, of some $sb$-later write). Finally, we say that an event $a$ happens-before another event $b$ if there is a path from $a$ to $b$ consisting of $sb$ and $sw$ edges.

Next, we define the extended coherence order, $eco$, to be $(rf \cup mo \cup rb)^+$, where $rb$ is the reads-before order. Since the modification order, $mo$, is transitive, the definition of $eco$ can be simplified as follows:

- $rb \triangleq rf^{-1}; mo$ (reads-before or from-read)
- $eco \triangleq rf \cup (mo \cup rb); rf^+$ (extended coherence order)

We remark that $eco$ is a strict order. Finally, we define the partial SC relation, $psc$, as follows.

- $psc \triangleq \left(\begin{array}{l} \text{ECO} \cup \text{ECO} \\ \text{HB} \cup \text{HB} \end{array}\right); \left(\begin{array}{l} sb \cup eco \cup \text{ECO} \\ sb|_{\text{local}}; sb|_{\text{local}}; sb|_{\text{local}} \end{array}\right); \left(\begin{array}{l} \text{ECO} \cup \text{ECO} \\ \text{HB} \cup \text{HB} \end{array}\right)$

Using these derived relations, RC11-consistency imposes four constraints on executions:

**Definition 1.** An execution $G$ is called **RC11-consistent** if it is complete and the following hold:
- $hb; eco^+$ is irreflexive. (**COHERENCE**)
- $rmw \cap (rb; mo) = \emptyset$. (**ATOMICITY**)
- $psc$ is acyclic. (**SC**)
- $sb \cup rf$ is acyclic. (**NO-THIN-AIR**)

**COHERENCE** ensures that programs with only one shared location are sequentially consistent, as at least two locations are needed for a cycle in $sb \cup eco$. **ATOMICITY** ensures that the read and the write comprising a RMW are adjacent in $eco$: there is no write event in between. The SC condition is the main novelty of RC11 and will be used to show the DRF-SC theorem and other criteria for ensuring sequential consistency (see §8). Finally, **NO-THIN-AIR** rules out thin-air behaviors, albeit at a performance cost, as we will see in §5.

### 3.3 Program Outcomes

Finally, in order to allow the compilation of non-atomic read and writes to plain machine load and store instructions (as well as the compiler to reorder such accesses), RC11 follows the “catch-fire” approach: races on non-atomic accesses result in undefined behavior, that is, any outcome is allowed. Formally, it is defined as follows.

**Definition 2.** Two events $a$ and $b$ are called **conflicting** in an execution $G$ if $a, b \in E$, $W \in \{\text{typ}(a), \text{typ}(b)\}$, $a \neq b$, and $1\text{oc}(a) = 1\text{oc}(b)$. A pair $(a, b)$ is called a race in $G$ (denoted $\langle a, b \rangle \in \text{race}$) if $a$ and $b$ are conflicting events in $G$, and $\langle a, b \rangle \notin hb \cup hb^{-1}$.

**Definition 3.** An execution $G$ is called **racy** if there is some $\langle a, b \rangle \in \text{race}$ with $na \in \{\text{mod}(a), \text{mod}(b)\}$. A program $P$ has **undefined behavior** under RC11 if it has some racy RC11-consistent execution.
3.4 Comparison with C11

Besides the new SC and NO-THIN-AIR conditions, RC11 differs in a few other ways from C11.

- It does not support consume accesses, a premise feature of C11 that is not implemented by major compilers, nor locks, as they can be straightforwardly implemented with release-acquire accesses.
- For simplicity, it assumes all locations are initialized.
- It incorporates the fixes proposed by Vafeiadis et al. [27], namely (i) the strengthening of the release sequences definition, (ii) the removal of restrictions about different threads in the definition of synchronization, and (iii) the lack of distinction between atomic and non-atomic locations (and accordingly omitting the problematic rf ⊆ hb condition for non-atomic locations). The third fix avoids out-of-thin-air problems that arise when performing non-atomic accesses to atomic location [5, §5].
- It does not consider “unsequenced races” between atomic accesses to have undefined behavior. (Such gratuitous undefined behavior is not needed for any of our results.)

We have also made three presentational changes: (1) we have a much more concise axiomatization of coherence; (2) we model RMWs using two events; and (3) we do not have a total order over SC atoms.

Proposition 1. RC11’s COHERENCE condition is equivalent to the conjunction of the following constraints of C11:

- hb is irreflexive. (IRREFLEXIVE-HB)
- rf; hb is irreflexive. (NO-FUTURE-READ)
- mo; rf; hb is irreflexive. (COHERENCE-RW)
- mo; hb is irreflexive. (COHERENCE-WW)
- mo; hb; rf −1 is irreflexive. (COHERENCE-RR)

Proposition 2. The SC condition is equivalent to requiring the existence of a total strict order S on En such that S; psc is irreflexive.

Finally, the next proposition ensures that without mixing SC and non-SC accesses to the same location, RC11 supplies the stronger guarantee of C11. As a consequence, programmers that never mix such accesses may completely ignore the difference between RC11 and C11 regarding SC.

Proposition 3. If all SC accesses are to distinguished locations (for every a, b ∈ E \ E0, if mod(a) = sc and loc(a) = loc(b) then mod(b) = sc) then [E±c]; hb; [E±c] ⊆ psc.
typically more SC reads than SC writes in programs, this scheme is less preferred.

5. Compilation to Power

In this section, we present the Power model and the mappings of language operations to Power instructions. We then prove the correctness of compilation from RC11 to Power.

As a model of the Power architecture, we use the recent declarative model by Algglave et al. [3], which we denote by Power. Its executions are similar to the ones above, with the following exceptions:

- Read/write labels have the form \( R(x, v) \) and \( W(x, v) \) (they do not include a “mode”).
- Power executions track syntactic dependencies between events in the same thread, and derive a relation called preserved program order, denoted \( ppo \), which is a subset of \( sb \) guaranteed to be preserved. The exact definition of \( ppo \) is quite intricate, and is included in Appendix F.
- Power has two types of fence events: a “lightweight fence” and a “full fence”. We denote by \( F^{lwsync} \) and \( F^{sync} \) the set of all lightweight fence and full fence events in a Power execution. Power’s “instruction fence” \( (iasync) \) is used to derive \( ppo \) but is not recorded in executions.

In addition to \( ppo \), the following additional derived relations are needed to define Power-consistency (see [3] for further explanations and details).

- \( sync \triangleq [Rw];sb;[F^{sync}];sb;[Rw] \)
- \( lwsync \triangleq [Rw];sb;[F^{lwsync}];sb;[Rw] \setminus (W \times R) \)
- \( fence \triangleq sync \cup lwsync \) (fence order)
- \( hb_p \triangleq ppo \cup fence \cup rfe \) (Power’s happens-before)
- \( prop_1 \triangleq [w];rfe';(fence;hb_p^*);[w] \)
- \( prop_2 \triangleq (mo(e_{urbe})')';rfe';(fence;hb_p^*);sync;hb_p^* \)
- \( prop \triangleq prop_1 \cup prop_2 \) (propagation relation)

where for every relation \( c \) (e.g., \( rf, mo \), etc.), we denote by \( ci \) and \( ce \) (internal and external) its thread-internal and thread-external restrictions. Formally, \( ci = c \cap sb \) and \( ce = c \setminus sb \).

Definition 5. A Power execution \( G \) is Power-consistent if it is complete and the following hold:

1. \( sb_l;loc \cup rf \cup rb \cup mo \) is acyclic. (SC-PER-LOC)
2. \( rbe;prop;hb_p^* \) is irreflexive. (OBSERVATION)
3. \( mo \cup prop \) is acyclic. (PROPAGATION)
4. \( rmw \land (rbe;mo) \) is irreflexive. (POWER-ATOMICITY)
5. \( hb_p \) is acyclic. (POWER-NO-THIN-AIR)

Remark 3. The model in [3] contains an additional constraint: \( mo \cup [At];sb;[At] \) should be acyclic (recall that \( At = dom(rmw) \cup codom(rmw) \)). Since none of our proofs requires this property, we excluded it from Def. 5.

\[
\begin{align*}
\langle R^{ac} \rangle & \triangleq 1d \quad \langle W^{ac} \rangle \triangleq st \\
\langle R^{lo} \rangle & \triangleq 1d;cmp;bc \quad \langle W^{lo} \rangle \triangleq st \\
\langle R^{acq} \rangle & \triangleq 1d;cmp;bc;isync \quad \langle W^{acq} \rangle \triangleq lwsync;st \\
\langle F^{loc} \rangle & \triangleq lwsync \quad \langle F^{acq} \rangle \triangleq sync \\
\langle R^{a}\rangle & \triangleq L;lwarx;cmp;bc Le;stwcx.;bc Le;:
\langle W^{a}\rangle \triangleq (\langle R^{a}\rangle;isync) \\
\langle R^{a}\rangle & \triangleq lwsync;\langle R\rangle^{a};isync \\
\langle R^{a}\rangle & \triangleq lwsync;\langle R\rangle^{a};isync
\end{align*}
\]

Figure 7. Compilation of non-SC primitives to Power.

Figure 8. Compilations of SC accesses to Power.

Unlike RC11, well-formed Power programs do not have undefined behavior. Thus, a function \( O : Loc \to Val \) is an outcome of a Power program \( P \) if it is an outcome of some Power-consistent full execution of \( P \) (see Def. 4).

As already mentioned, the two compilation schemes from C11 to Power that have been proposed in the literature [1] differ only in the mappings used for SC accesses (see Fig. 8). The first compilation scheme follows the leading sync convention, and places a sync fence before each SC access. The alternative scheme follows the trailing sync convention, and places a sync fence after each SC access. Importantly, the same scheme should be used for all SC accesses in the program, since mixing the schemes is unsound. The mappings for the non-SC accesses and fences are common to both schemes and are shown in Fig. 7. Note that our compilation of relaxed reads is stronger than the one proposed for C11 (see §2.4).

Our main theorem says that the compilation schemes are correct. For a program \( P \), we denote by \( \langle P \rangle \) the Power program obtained by compiling \( P \) using the scheme in Fig. 7 and either of the schemes in Fig. 8 for SC accesses.

Theorem 1. Given a program \( P \), every outcome of \( \langle P \rangle \) under Power is an outcome of \( P \) under RC11.

Proof (Outline). The main idea is to consider the compilation as if it happens in three steps, and prove the soundness of each step:

1. Leading sync: Each \( R^{ac} / W^{ac} / RMW^{ac} \) in \( P \) is replaced by \( F^{ac} \) followed by \( R^{a};rbe;mo \) \( / RMW^{acrel} \).

Trailing sync: Each \( R^{ac} / W^{ac} / RMW^{ac} \) in \( P \) is replaced by \( R^{a};rbe;mo \) \( / RMW^{acrel} \) followed by \( F^{ac} \).

2. The mappings in Fig. 7 are applied.

3. Leading sync: Pairs of the form \( sync; lwsync \) that originated from \( R^{ac} / W^{ac} / RMW^{ac} \) are reduced to \( sync \) (eliminating the redundant \( lwsync \)).

Trailing sync: Any \( cmp;bc;isync;sync \) sequences originated from \( R^{ac} / W^{ac} / RMW^{ac} \) are reduced to \( sync \) (eliminating the redundant \( cmp;bc;isync \)).
The resulting Power program is clearly identical to the one obtained by applying the mappings in Figures 7 and 8.

Soundness of the steps (that is, none of them introduces additional outcomes) follows from Lemmas E.2 and H.2 and Appendix F.3.

The main difficulty (and novelty of our proof) lies in proving soundness of the second step, and more specifically in establishing the NO-THIN-AIR condition. Since Power, unlike RC11, does not generally forbid \( \text{sb} \cup \text{rf} \) cycles, we have to show that such cycles can be untangled to produce a racy RC11-consistent execution, witnessing the undefined behavior. Here, the idea is, similar to DRF-SC proofs, to detect a first \( \text{rf} \)-edge that closes an \( \text{sb} \cup \text{rf} \) cycle, and replace it by a different \( \text{rf} \)-edge that avoids the cycle. This is highly non-trivial because it is unclear how to define a “first” \( \text{rf} \)-edge when \( \text{sb} \cup \text{rf} \) is cyclic. To solve this problem, we came up with a different ordering of events, which does not include all \( \text{sb} \) edges, and Power ensures to be acyclic (a relation we call Power-before in Appendix G).

For completeness, we also show that the conditional branch after the relaxed read is only necessary if we care about enforcing the NO-THIN-AIR condition. That is, let weakRC11 be the model obtained from RC11 by omitting the NO-THIN-AIR condition, and denote by \( (P)_{\text{weak}} \) the Power-program obtained by compiling \( P \) as above, except that relaxed reads are compiled to plain loads (again, with either leading or trailing syncs for SC accesses). Then, this scheme is correct with respect to the weakRC11 model.

Theorem 2 (Compilation of weakRC11 to Power). Given a program \( P \), every outcome of \( (P)_{\text{weak}} \) under Power is an outcome of \( P \) under weakRC11.

Finally, we note that it is also possible to use a lightweight fence (\( \text{lwsync} \)) instead of a fake control dependency and an instruction fence (\( \text{lsync} \)) in the compilation of (all or some) acquire accesses. The correctness follows from Appendix F.3.

6. Compilation to ARMv7

The ARMv7 model [3] is very similar to the Power model just presented in §5. There are only two differences.

First, while ARMv7 has analogues for Power’s strong fence (\( \text{sync} \)) and instruction fence (\( \text{lsync} \)), it lacks an analogue for Power’s lightweight fence (\( \text{lwsync} \)). Thus, on ARMv7 we have \( \text{f}^{\text{lwsync}} = \emptyset \) and so \( \text{fence} = \text{sync} \).

The second difference is that ARMv7 has a somewhat weaker preserved program order, ppo, than Power, which in particular does not always include \( [\text{rb}] ; \text{sb} ; [\text{lw}] \) (following the model in [3]). In our Power compilation proofs, however, we never rely on this property of Power’s ppo (see Appendix F).

The compilation schemes to ARMv7 are essentially the same as those to Power substituting the corresponding ARMv7 instructions for the Power ones: \( \text{dmb} \) instead of \( \text{sync} \) and \( \text{lwsync} \), and \( \text{lb} \) instead of \( \text{lsync} \). (Since ARMv7 lacks an analogue for \( \text{lwsync} \), the compilation to ARMv7 uses a strong fence (\( \text{dmb} \)) instead.) The soundness of compilation to ARMv7 follows directly from Theorems 1 and 2.

We note that neither GCC (version 5.4) nor LLVM (version 3.9) map acquire reads into \( 1d; \text{cmp}; \text{bc}; \text{lb} \). Instead, they emit \( 1d; \text{dmb} \) (that corresponds to Power’s \( 1d; \text{sync} \)). With this stronger compilation scheme, there is no correctness problem in compilation of C11 to ARMv7. Nevertheless, if one intends to use \( \text{lb} \)’s, the same correctness issue arises (e.g., the one in Fig. 1), and RC11 overcomes this issue.

7. Correctness of Program Transformations

In this section, we list program transformations that are sound in RC11, and prove that this is the case. As in [27], to have a simple presentation, all of our arguments are performed at the semantic level, as if the transformations were applied to events in an execution. Thus, to prove soundness of a program transformation \( P_{\text{src}} \rightsquigarrow P_{\text{tgt}} \), we are given an arbitrary RC11-consistent execution \( G_{\text{tgt}} \) of \( P_{\text{tgt}} \), and construct a RC11-consistent execution \( G_{\text{src}} \) of \( P_{\text{src}} \), such that either \( G_{\text{src}} \) and \( G_{\text{tgt}} \) have the same outcome or \( G_{\text{src}} \) is racy. In the former case, we show that \( G_{\text{tgt}} \) is racy only if \( G_{\text{src}} \) is. Consequently, one obtains that every outcome of \( P_{\text{tgt}} \) under RC11 is also an outcome of \( P_{\text{src}} \) under RC11.

The soundness proofs (sketched in Appendix I) are mostly similar to the proofs in [27], with the main difference concerning the new SC.

**Strengthening** Strengthening transforms the mode \( o \) of an event in the source into \( o' \) in the target where \( o \sqsubseteq o' \). Soundness of this transformation is trivial, because RC11-consistency is monotone with respect to the mode ordering.

**Sequentialization** Sequentialization merges two program threads into one, by interleaving their events in \( \text{sb} \). Essentially sequentialization just adds edges to the \( \text{sb} \) relation.
soundness trivially follows from the monotonicity of RC11-consistency with respect to sb.

**Deordering** Table 1 defines the deorderable pairs, for which we proved the soundness of the transformation X; Y ↦ Y in RC11. (Note that deordering is obtained by applying deordering and sequentialization.) Generally speaking, RC11 supports all reorderings that are intended to be sound in C11 [27], except for load-store reorderings of relaxed accesses, which are unsound in RC11 due to the conservative NO-THIN-AIR condition (if one omits this condition, these reorderings are sound). Importantly, load-store reorderings of non-atomic accesses are sound due to the “catch-fire” semantics. The soundness of these reorderings (in the presence of NO-THIN-AIR) was left open in [27], and requires a non-trivial argument of the same nature as the one used to show NO-THIN-AIR in the compilation correctness proof (see Appendices G and I).

**Merging** Merges are transformations of the form X; Y ↦ Z, eliminating one memory access or fence. Fig. 9 defines the set of mergeable pairs. Note that using strengthening, the modes mentioned in Fig. 9 are upper bounds (e.g., \(R^{acq}; R^{rlx}\) can be first strengthened to \(R^{acq}; R^{rlx}\) and then merged). Generally speaking, RC11 supports all mergings that are intended to be mergeable in C11 [27].

**Remark 4.** The elimination of redundant read-after-write allows the write to be non-atomic. Nevertheless, an SC read cannot be eliminated in this case, unless it follows an SC write. Indeed, elimination of an SC read after a non-SC write is unsound in RC11. We note that while this elimination is allowed by a certain fix of C11 described in [27], its effectiveness seems to be low, and, in fact, it is already unsound for the model in [4] (see Appendix A.3 for a counterexample). Note also that read-after-RMW elimination does not allow the read to be an acquire read unless the update includes an acquire read (unlike read-after-write). This is due to release sequences: eliminating an acquire read after a relaxed update may remove the synchronization due to a release sequence ending in this update.

**Register Promotion** Finally, “register promotion” is sound in RC11. This global program transformation replaces all the accesses to a memory location by those to a register, provided that the location is used by only one thread. At the execution level, all accesses to a particular location are removed from the execution, provided that they are all sb-related.

### Table 1: Deorderable pairs

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(R^o; R^o)</td>
<td>(\rightarrow R^o)</td>
</tr>
<tr>
<td>(W^{rb}; R^{sc})</td>
<td>(\rightarrow W^{rb})</td>
</tr>
<tr>
<td>(RMW^{rb}; R^{sc})</td>
<td>(\rightarrow RMW^{rb})</td>
</tr>
<tr>
<td>(W^{rb}; RMW^{rb})</td>
<td>(\rightarrow W^{rb})</td>
</tr>
<tr>
<td>(F^o; F^o)</td>
<td>(\rightarrow F^o)</td>
</tr>
</tbody>
</table>

8. Programming Guarantees

In this section, we demonstrate that our semantics for SC atomics (i.e., the SC condition in Def. 1) is not overly weak. We do so by proving theorems stating that programmers who follow certain defensive programming patterns can be assured that their programs exhibit no weak behaviors. The first such theorem is DRF-SC, which says that if a program has no races on non-SC accesses under SC semantics, then its outcomes under RC11 coincide with those under SC.

In our proofs we use the standard declarative definition of SC: an execution is SC-consistent if it is complete, satisfies ATOMICITY, and \(sb \cup rf \cup mo \cup rb\) is acyclic [25].

**Theorem 3.** If in all SC-consistent executions of a program \(P\), every race \(\langle a, b \rangle\) has \(mod(a) = mod(b) = \text{sc}\), then the outcomes of \(P\) under RC11 coincide with those under SC.

Next, we show that adding a fence instruction between every two accesses to shared locations restores SC, or there remains a race in the program, in which case the program has undefined behavior. More formally, we say that a location is shared if it is accessed by more than one thread.

**Definition 6.** A location \(x\) is shared in an execution \(G\) if \(\langle a, b \rangle \notin sb \cup sb^{-1}\) for some distinct events \(a, b \in E_G\).

**Theorem 4.** Let \(G\) be an RC11-consistent execution. Suppose that for every two distinct shared locations \(x\) and \(y\), \([E_G^x; sb; E_G^y] \subseteq sb; [F_E^x]; sb\). Then, \(G\) is SC-consistent.

We remark that for the proofs of Theorems 3 and 4, we do not need the full SC condition: for Thm. 3 it suffices for \([E_E^x]; sb; Eco; sb; [F_E^x]; sb\) to be acyclic; and for Thm. 4 it suffices for \([F_E^x]; sb; Eco; sb; [F_E^x]\) to be acyclic.

9. Related Work

Despite having been developed quite recently, a fair number of problems have been found in the C11 memory model. The model itself was designed by the C++ standard committee based on a paper by Boehm and Adve [9]. During the standardization process, Batty et al. [7] formalized the C11 memory model and proved soundness of its compilation to x86-TSO. They also proposed a number of key technical improvements to the model (such as some coherence axioms), which were incorporated into the standard.

Soon afterwards, Batty et al. [6] and Sarkar et al. [24] studied the compilation of C11 to Power, and incorrectly proved the correctness of two compilation schemes. In their proofs, from a consistent Power execution, they constructed a corresponding C11 execution, which they tried to prove consistent, but in doing so they forgot to check the overly strong condition S1. The examples shown in the introduction and in §2.1 are counterexamples to their theorems.

Quite early on, a number of papers [11, 28, 22, 10] noticed the disastrous effects of thin-air behaviors allowed by the C11 model, and proposed strengthening the definition of consistency by disallowing \(sb \cup rf\) cycles. [10] further
discussed how the compilation schemes of relaxed accesses to Power and ARM would be affected by the change, but did not formally prove the correctness of their proposed schemes.

Next, Vafeiadis et al. [27] noticed a number of other problems with the C11 memory model, which invalidated a number of source-to-source program transformations that were assumed to hold. They proposed local fixes to those problems, and showed that these fixes enabled proving correctness of a number of local transformations. We have incorporated their fixes in the RC11-consistency definition.

Then, in 2016, Batty et al. [4] proposed a more concise semantics for SC atomics, whose presentation we have followed in our proposed RC11 model. As their semantics is stronger than C11, it cannot be compiled efficiently to Power, contradicting the claim of that paper. Moreover, as already discussed, SC fences are still too weak according to their model: in particular, putting them between every two accesses in a program with only atomic accesses does not guarantee SC.

Finally, Manerkar et al. [20] recently discovered the problem with trailing-sync compilation to Power (in particular, they observed the IRIW-acq-sc counterexample), and identified the mistake in the existing proof. Independently, we discovered the same problem, as well as the problem with leading-sync compilation. Moreover, in this paper, we have proposed a fix for both problems, and proven that it works.

A number of works [28, 26, 16, 15] have previously studied only small fragments of the C11 model—typically the release/acquire fragment. Among these, Lahav et al. [15] previously proposed strengthening the semantics of SC fences in a different way by treating them as read-modify-writes to a distinguished location. That strengthening, however, was considered in the restricted setting of only release/acquire accesses, and does not directly scale to the full set of C11 access modes. In fact, for the fragment containing only SC fences and release/acquire accesses, RC11-consistency is equivalent to RA-consistency that treats SC fences as RMWs to a distinguished location [15].

10. Conclusion

In this paper, we have introduced the RC11 memory model, which corrects all the known problems of the C11 model. We have further proved (i) the correctness of compilation from RC11 to x86-TSO [23], Power and ARMv7 [3]; (ii) the soundness of various program transformations; (iii) a DRF-SC theorem; and (iv) a theorem showing that for programs without non-atomic accesses, weak behaviors can be always avoided by placing SC fences. It would be useful to mechanize the proofs of this paper in a theorem prover; we leave this for future work.

A certain degree of freedom exists in the design of the SC condition. A very weak version, which still maintains the two formal programming guarantees of this paper, would require acyclicity of \((E^2 \cup E^c) : sc\); \((sc \cup \text{eco})\); \((E^c) \cup sc\); \((F^c)\)). In our choice of \(psc\) we aimed to provide: (i) stronger guarantees, while still, needless to say, ensuring the correctness of compilation; and (ii) additional optimization opportunities which are sound for sequential consistency (eliminating overwritten SC writes and repeated SC reads).

Regarding the infamous out-of-thin-air problem, we employed in RC11 a conservative solution at the cost of including a fake control dependency after every relaxed read. While this was already considered as a valid solution before, we are the first to prove the correctness of this compilation scheme, as well as the soundness of reordering of independent non-atomic accesses under this model. Correctness of an alternative scheme that places a lightweight fence after every relaxed write is left for future work. It is interesting to check the practical performance costs of each scheme. On the one hand, relaxed writes (which are not followed by a fence) are perhaps rare in real programs, compared to relaxed reads. On the other hand, a control dependency is cheaper than a lightweight fence, and relaxed reads are often anyway followed by a control dependency.

Another important future direction would be to combine our SC constraint with the recent model of Kang et al. [14], which prevents out-of-thin-air values (and avoids undefined behaviors all together), while still allowing the compilation of relaxed reads and writes to plain loads and stores. This is, in particular, crucial for adopting a model like RC11 in a type-safe language, like Java, that cannot allow undefined behaviors. Integrating our SC condition in that model, however, is non-trivial because the Kang et al. [14] model is defined in a very different style from C11, and thus we will have to find an equivalent operational way to check our SC condition.

Finally, establishing the correctness of compilation of RC11 to ARMv8 [12] is another important future goal.

Acknowledgments

This research was supported in part by an ERC Consolidator Grant for the project “RustBelt” (grant agreement no. 683289), and in part by Samsung Research Funding Center of Samsung Electronics under Project Number SRFC-IT1502-07. The third author has been supported by a Korea Foundation for Advanced Studies Scholarship.

References


A. Further Examples

A.1 Failure of leading sync convention with SC fences

The following behavior is disallowed according to C11, but
allowed by its compilation to Power.

\[
\begin{align*}
  x & := r_{lx} 2 \\
fence_{sc} & := y_{rlx} 0 \\
b & := y_{rlx} / 0 \\
y & := s_{c} 1 \\
d & := x_{r_{lx}} / 2 \\
e & := x_{ac} \# 1
\end{align*}
\]

(\(R_{sync}+R_{sc}\))

Under C11, this behavior is forbidden. Consider the following
execution (the initialization of \(x\) is omitted):

\[
\begin{align*}
  k & : W_{l_{x}}(x, 2) \\
l & : F_{sc}(x, 1) \\
m & : W_{r}(y, 0) \\
p & : W_{l_{x}}(x, 2) \\
q & : R_{e_{x}}(x, 1)
\end{align*}
\]

The \(rf\) and \(mo\) edges are forced because of coherence.
Now, the C11 conditions on SC fences require, in particular,
that \([F_{sc}]; s_{b}; r_{f}; [E_{sc}]; r_{f}; s_{b}; [F_{sc}] \subseteq S\). Hence, we must have \(S(l, n)\) (essentially because if we had
\(S(n, l)\), then \(m\) would have been reading from an overwritten
write), as well as \(S(q, l)\) (essentially because if we had \(S(l, q)\),
then \(m\) would have been reading from an \(mo\)-overwritten
write before the fence). By transitivity, we thus have \(S(q, n)\)
which contradicts condition S1, which requires \(S(n, q)\) be-
cause of the happens-before path via \(o\).

The compilation to Power allows the behavior because
again the sync fences do not provide sufficient synchronization:
again all but one sync fences are useless, as they are
placed at the beginning of a thread.

A.2 Failure of write-after-write and read-after-read

elminations using \(psc_{3}\)

We present the executions showing the failure of write-after-
write and read-after-read eliminations using \(psc_{3}\) (see \S2.3).

First, the following execution is an execution of \(WW_{merge}\)
yielding the result \(a = 2 \& b = c = 0\). The initialization events
are omitted.

\[
\begin{align*}
  k & : R_{e_{x}}(x, 2) \\
m & : W_{e_{x}}(x, 1) \\
l & : R_{e_{x}}(y, 0) \\
r & : R_{e_{x}}(x, 0) \\
o & : W_{e_{x}}(y, 1)
\end{align*}
\]

This execution is inconsistent with the SC condition
that requires acyclicity of \(psc_{3}\) (since we have \(⟨m, l⟩, ⟨l, o⟩, ⟨o, p⟩, ⟨p, m⟩ \in psc_{3}\)). It is, however, consistent
using our final \(psc\) relation \(⟨m, l⟩ \notin psc\).

Now, the following execution is an execution of the same
\(WW_{merge}\) program, but after applying the elimination of \(x := s_{c} 1\),
again yielding the result \(a = 2 \& b = c = 0\). This execution is consistent with the SC condition that requires
acyclicity of \(psc_{3}\) (as well as with our final \(psc\) relation).

A.3 Failure of SC-read-after-non-SC-write elimination

\[
\begin{align*}
  y_{sc} & := 2; \\
x & := r_{lx} 1; \\
y & := s_{c} 2; \\
\end{align*}
\]

\[
\begin{align*}
  x & := r_{lx} 1; \\
y & := s_{c} 2; \\
\end{align*}
\]

\[
\begin{align*}
  a & := x_{ac}; \\
c & := y_{rlx}; \\
b & := x_{r_{lx}}; \\
\end{align*}
\]

The annotated behavior is allowed under RC11 for the
target, but not for the source. The same applies to the model of
Batty et al. [4].
B. Programs to Executions: Receptiveness Assumption

To carry out the compilation correctness proof, we need to record syntactic dependencies between instructions, as in the Power model. (This is only needed if one is interested in the NO-THIN-AIR condition; compilation correctness for weakRC11 may completely ignore this extension.) Dependencies are classified into data, address and control dependencies. Accordingly, we extend the definition of an execution (see §3.1), with additional relations data, addr and ctrl. We use deps to denote the union of the three relations. We require data, addr and ctrl to satisfy the following:

1. \( \text{data} \subseteq R \times W \).
2. \( \text{addr} \subseteq R \times (R \cup W) \).
3. \( \text{ctrl} \subseteq R \times E \).
4. \( \text{ctrl; sb} \subseteq \text{ctrl} \).
5. \( \text{rmw} \subseteq \text{deps} \).

The dependency relations are calculated from the program syntax, together with the generation of program’s execution (like in Power), and the construction ensures that the above properties hold. Moreover, the construction of executions from programs provides us with the following receptiveness property:

**Definition B.1.** A function \( \text{lab} \) : Event \( \rightarrow \) Label is called a reevaluation of \( \text{lab} \) : Event \( \rightarrow \) Label if for every event \( a \), the label \( \text{lab}(a) \) is identical to \( \text{lab}(a) \), except possibly for read/written value.

**Notation B.1.** Given an execution \( G \) and a reevaluation \( \text{lab} \) of \( G \), \( \text{lab}(G) \) denotes the execution \( G' \) given by: \( G'.\text{lab} = \text{lab} \), \( G'.\text{rf} = \emptyset \), and \( G'.c = G.c \) for every \( c \in \{ E, sb, rmw, mo, data, addr, ctrl \} \).

**Assumption B.1** (receptiveness). Let \( G \) be an execution of a program \( P \). Let \( a \in R \), and suppose that \( a \notin \text{dom}(\text{deps}; (\text{ctrl} \cup \text{addr})) \). For every \( v \in \text{Val} \), there exists a reevaluation \( \text{lab} \) of \( G \) such that:

- \( \text{lab}(G) \) is an execution of \( P \).
- \( \text{lab}(G).\text{val}(a) = v \).
- \( \text{lab}(b) = G.\text{lab}(b) \) whenever \( \langle a, b \rangle \notin G.\text{deps}^+ \).

Note that a more basic receptiveness property follows from this assumption: if \( a \notin \text{dom}(\text{sb}) \) then for every \( v \in \text{Val} \), we have that \( \text{lab}(G) \) is an execution of \( P \), for the reevaluation \( \text{lab} \) of \( G \) that sets the read value of \( a \) to \( v \), and otherwise is identical to \( G.\text{lab} \).

In addition, we assume that the set of executions of a program is prefix-closed:

**Notation B.2.** Given an execution \( G \) and a set \( E \subseteq E \) that is downwards closed w.r.t. \( \text{sb} \) (i.e., \( a \in E \) whenever \( \langle a, b \rangle \in \text{sb} \) for some \( b \in E \)), and contains at least all the initialization events, the restriction of \( G \) to \( E \), denoted \( G|_E \), is the execution \( G' \) given by \( G'.E = E \), \( G'.\text{lab} = G.\text{lab}|_E \), and \( G'.c = [E]; G.c; [E] \) for \( c \in \{ \text{sb}, \text{rmw}, \text{rf}, \text{mo}, \text{data}, \text{addr}, \text{ctrl} \} \).

**Assumption B.2** (prefix-closed executions). Let \( G \) be an execution of a program \( P \), and let \( E \) be a subset of \( E \) that is downwards closed w.r.t. \( \text{sb} \), and contains at least all the initialization events. Then, \( G|_E \) is an execution of \( P \).

C. Properties of RC11

In this section, we present some basic properties of the derived relations \( \text{eco}, \text{sw}, \text{hb} \) and of RC11-consistent executions. We omit some of the proofs that straightforwardly follow from our definitions. For the rest of this section, consider an arbitrary execution \( G \).

**Proposition C.1.** \( \text{eco} \) is a strict partial order.

**Proposition C.2.** Suppose that \( [W]; \text{sb}|_{\text{loc}}; [W] \subseteq \text{mo} \) and \( \text{rmw} \subseteq \text{rb} \). Then, the following hold:

1. \( \text{rs} \subseteq \text{eco}^* \).
2. \( [W]; \text{sw}; [R] \subseteq \text{eco} \).
3. \( [W]; \text{sw}; [F] \subseteq \text{eco}; \text{sb} \).
4. \( \text{eco}; \text{hb} \subseteq \text{eco} \cup \text{eco}; (\text{sb} \setminus \text{rmw}); \text{hb}^* \).

**Proof.**

1. Let \( \langle a, b \rangle \in \text{rs} \). Then, by definition, \( \langle a, b \rangle \in [W]; \text{sb}|^2_{\text{loc}}; [W]^{\geq \text{rf}x}; (\text{rf}; \text{rmw})^* \). Since \( [W]; \text{sb}|_{\text{loc}}; [W] \subseteq \text{mo} \) and \( \text{rmw} \subseteq \text{rb} \), we have \( \langle a, b \rangle \in \text{eco}^* \). Since \( \text{eco} \) is transitive, we have \( \langle a, b \rangle \in \text{eco}^* \).
2. Let \((a, b) \in |W|; sw; [R]\). Then, by definition, we have \((a, b) \in rs; rf\). Using the previous item, we obtain that \((a, b) \in eco''; eco \subseteq eco\).

3. Let \((a, b) \in |W|; sv; [F]\). Then, by definition, we have \((a, b) \in rs; rb; sb\). Using the first item, we obtain that \((a, b) \in eco''; eco \subseteq eco; sb\).

4. Let \((a, c) \in eco; hb\), and let \(b \in E\) be an eco-maximal event satisfying \((a, b) \in eco\), and \((b, c) \in hb\). If \(b = c\) then \((a, c) \in eco\), and we are done. Otherwise, the maximality of \(b\) ensures that \((b, b') \in sb \setminus sv\) and \((b', c) \in hb\) for some \(b' \in E\). Since \(rmw \subseteq rb \subseteq eco\), it follows that \((a, c) \in eco; (sb \setminus rmw); hb\).

**Lemma C.1** (Read at end). Let \(a \in R \setminus dom(sb)\). Suppose that \(G' = G|_{G\setminus\{a\}}\) is RC11-consistent. Then, there exists an event \(b \in G'\cdot W\) such that the execution \(G''\) given by \(G''\cdot c = G\cdot c\) for every \(c \in \{E, sb, rmw, data, addr, ctrl, mo\}\), \(G''\cdot lab = G'\cdot lab \cup \{a \mapsto r\mod(a)\cdot(loc(a), val_e(b))\}\), and \(G''\cdot rf = G'\cdot rf \cup \{\langle a, b \rangle\}\) is RC11-consistent.

**Proof.** Take \(b\) to be the max-maximal event in \(G\cdot W_{loc(a)}\). It is straightforward to show that \(G''\), as defined in the statement, is RC11-consistent.

**Proposition C.3.** Let \(a \in W^{E=\{\cdot\} \setminus dom(rf)}\). Let \(G' = G|_{G\setminus\{a\}}\). Then, \([G'.E]; G.hb; [G'.E] = G'.hb\).

**Proposition C.4.** Let \(G'\) be any execution obtained from \(G\) by possibly changing the value read at some \(a \in R^{na}\), and the source of the \(rf\) edge entering the event \(a\). Then, \(G'.hb = G.hb\).

**Proposition C.5.** Let \(G'\) be an execution, such that \(G'.E = G.E \cup \{a\}\) for some event \(a\). Suppose that \(a \in G'.R^{na}, G.sb \subseteq G'.sb, G.lab \subseteq G'.lab, G.rmw = G'.rmw, \) and \(G'.rf = G.rf \cup \{\langle b, a \rangle\}\) for some \(b \in G.E\). Then, \([G.E]; G'.hb; [G.E] = G.hb\).

### D. The RC\textsubscript{na} Model

In this section we present a variant of RC11, which has a smaller psc relation, and is useful in our correctness of compilation proofs. It is based on the following additional derived relations:

\[
\begin{align*}
rb^{na} & \triangleq [R^{na}]; rb \\
rb^{\neq na} & \triangleq rb \setminus rb^{na} \\
eco^{\neq na} & \triangleq rf \cup \{mo \cup rb^{\neq na}; rf\} \\
psc^{\neq na} & \triangleq ([E^{sc}] \cup [F^{sc}]; hb); (sb \cup eco^{\neq na} \cup sb\cdot|_{loc}; hb; sb\cdot|_{loc}); ([E^{sc}] \cup hb; [F^{sc}])
\end{align*}
\]

**Proposition D.1.** If \(rb^{na} \subseteq hb\) then \(psc = psc^{\neq na}\).

**Proof.** Immediately follows from our definitions. (Note that \(psc \setminus psc^{\neq na} \subseteq [F^{sc}]; hb; (rb^{\neq na}; rf); ([E^{sc}] \cup hb; [F^{sc}]), \) and \(hb; rf \subseteq hb'); (sb \cup rf).)

We call an execution RC\textsubscript{na}-consistent if it satisfies all conditions of Def. 1, except possibly for sc, and psc\textsuperscript{\neq na} is acyclic.

**Lemma D.1.** Let \(G\) be an RC\textsubscript{na}-consistent execution of a program \(P\). Then, either \(G\) is RC11-consistent, or \(P\) has undefined behavior under RC11.

**Proof.** If \(rb^{na} \subseteq hb\), then, by Prop. D.1, \(psc = psc^{\neq na}\) and \(G\) is RC11-consistent. Suppose otherwise. We show that \(P\) has undefined behavior under RC11. Let \(a_1, \ldots, a_n\) be an enumeration of \(E\) that respects \(sb \cup rf\) (that is, \(i < j\) whenever \((a_i, a_j) \in sb \cup rf\)). For every \(1 \leq i \leq n\), let \(E_i = E_0 \cup \{a_1, \ldots, a_i\}\) and \(G_i = G|_{E_i}\). Let \(k\) be the minimal index such that \(G_k.rb^{na} \not\subseteq G_k.hb\). Then, by Prop. D.1, \(G_{k-1}.psc = G_{k-1}.psc^{\neq na}\) is acyclic, and so \(G_{k-1}\) is RC11-consistent. Let \((a_k, a_k) \in G_k.rb^{na} \setminus G_k.hb\). Then, we must have \(a_k \in \{a_k, a_k\}\). Note also that \((a_k, a_k) \notin G_k.hb\) since \(G_k\) satisfies COHERENCE.

Now, if \(G_k\) is RC11-consistent, then we are done (it is a racy execution of \(P\)). Suppose otherwise. We show that \(a_k \neq a_k\). Indeed, since \(G_k\) is RC\textsubscript{na}-consistent but not RC11-consistent, and \(G_{k-1}\) is RC11-consistent, it must be the case that \(mod(a_k) = sc\), and there exist \(b, f \in E_{k-1}\) such that:

- \((a_k, b) \in G_k.mc; G_{k-1}.rf; (G_{k-1}.hb; [F]); (G_{k-1}.E^{sc})\)
Now, since we have $[E_{k-1}]; G_{k}.rb; G_{k}.mo; [E_{k-1}] \subseteq G_{k-1}.rb$, it follows that $(f, b) \in G_{k-1}.psc$. This, however, contradicts the fact that $G_{k-1}$ is RC11-consistent.

Therefore, we have $a_k = a_k$. Let $x = G.loc(a_k)$. By Lemma C.1, there exists an event $b \in G_{k-1}.w_x$ such that the execution $G'$ given by $G'.c = G.k.c$ for every $c \in \{E, sb, rmw, data, addr, ctrl, mo\}$, $G'.lab = G_{k-1}.lab \cup \{a_k \mapsto R^{na}(x, val_a(b))\}$, and $G'.rf = G_{k-1}.rf \cup \{(b, a_k)\}$ is RC11-consistent. By Assumption B.1, $G'$ is an execution of $P$. In addition, we have $\langle a_k, a_k \rangle \notin G'.hb$ (since $\langle a_k, a_k \rangle \notin G_{k}.hb$ and $G'.hb = G_{k}.hb$ by Prop. C.4), and so $G'$ is racy. Hence, $P$ has undefined behavior under RC11.

Next, we prove some lemmas that allow us (under some restrictions) to add a memory access inside a given execution. In what follows, we take $G$ to be an arbitrary execution.

Proposition D.2. If $a \notin dom(sb^2; [E^{rel}])$, then for every $b \in E$, we have $\langle a, b \rangle \in hb$ iff $\langle a, b \rangle \in sb$.

Proof. The assumption that $a \notin dom(sb^2; [E^{rel}])$ ensures that $a \notin dom(sb^2; sw)$, and so we have $\langle a, b \rangle \in hb$ iff $\langle a, b \rangle \in sb$.

Lemma D.2 (Add write). Let $a \in W \setminus \{\text{dom}(sb^2; [E^{rel}]) \cup \text{At}\}$. Suppose that $G' = [G; E, \{a\}]$ is RC$_{na}$-consistent. Let $x = \text{loc}(a)$, and suppose that $(a, b) \in sb; [R_x]$ implies $(a, b) \in sb; [W_x]; sb$. Then, there exists a relation $T \subseteq G.W_x \times G.W_x$ such that the execution $G''$ given by $G''.c = G.c$ for every $c \in \{E, lab, sb, rmw, data, addr, ctrl\}$, $G''.rf = G'.rf$, and $G''.mo = G'.mo \cup T$ is RC$_{na}$-consistent.

Proof. Let $C = \{c \in G'.W_x \mid \langle a, c \rangle \in G; sb; G'.mo\}$, and take $T = ((a) \times C) \cup ((G'.W_x \setminus C) \times \{a\})$. It is straightforward to show that $G''$, as defined in the statement, is RC$_{na}$-consistent. In particular, we have $G''.psc^{\neq na} = G'.psc^{\neq na}$.

Lemma D.3 (Add rmw write). Suppose that $\text{rmw}^{-1}; \text{rf}^{-1}; \text{rf}; \text{rmw} \subseteq [G; E]$. Let $a \in (W \setminus \text{At}) \setminus \text{dom}(sb^2; [E^{rel}])$. Suppose that $G' = [G; E, \{a\}]$ is RC$_{na}$-consistent. Let $x = \text{loc}(a)$, and suppose that $(a, b) \in sb; [R_x]$ implies $(a, b) \in sb; [W_x]; sb$. Then, there exists a relation $T \subseteq G.W_x \times G.W_x$ such that the execution $G''$ given by $G''.c = G.c$ for every $c \in \{E, lab, sb, rmw, data, addr, ctrl\}$, $G''.rf = G'.rf$, and $G''.mo = G'.mo \cup T$ is RC$_{na}$-consistent.

Proof. Let $b, d \in G'.E$ such that $(b, a) \in G; rmw$ and $(d, b) \in G'.rf$. Let $C = \{c \in G'.W_x \mid \langle a, c \rangle \in G'.mo\}$, and take $T = ((a) \times C) \cup ((G'.W_x \setminus C) \times \{a\})$. It is straightforward to show that $G''$, as defined in the statement, is RC$_{na}$-consistent.

Lemma D.4 (Add non-atomic read). Let $a \in R^n \setminus \text{dom}(sb; [E^{rel}])$. Suppose that $G' = [G; E, \{a\}]$ is RC$_{na}$-consistent. Then, there exists an event $b \in G'.W$ such that the execution $G''$ given by $G''.E = G.E$,

$G''.lab = G'.lab \cup \{a \mapsto R^{na}(\text{loc}(a), \text{val}_a(b))\}$,

$G''.c = G.c$ for every $c \in \{sb, rmw, data, addr, ctrl\}$, $G''.rf = G'.rf \cup \{(b, a)\}$, and $G''.mo = G'.mo$ is RC$_{na}$-consistent.

Proof. Let $x = \text{loc}(a)$. Let $B = \{b \in G; W_x \mid \langle b, a \rangle \in G; \text{rf}^2; G; \text{hb}\}$, and take $b$ be the mo-maximal event in $B$. It is straightforward to show that $G''$, as defined in the statement, is RC$_{na}$-consistent.

E. Proof of Global Transformation of SC accesses

In this section we prove the soundness of a global program transformation that either adds an SC fence before every SC access, or adds an SC fence after every SC access, and then replaces all SC accesses by release/acquire ones. This will allow us later to prove the correctness of compilation only for programs that do not contain any SC accesses.

We use the following additional notations:

$$psc_f \triangleq [F]; psc; [F]$$

$$sb' \triangleq sb \setminus \text{rmw}$$
Lemma E.1. Let $G$ be an execution satisfying all conditions of Def. 1, except possibly for $SC$. Suppose that $[RW^{sc}]; sb'; \cup sb'; hb; sb'; [RW^{sc}]; [F^sc]; hb$. Let $T = sb \cup sb; \neq loc; hb; sb; \neq loc \cup eco$. Then:

1. $[F^{sc}]; hb; eco'; ([RW^{sc}]; T; [RW^{sc}])^*; eco'; hb; [F^{sc}] \subseteq psc^{+}_{F}$.

2. If $psc_{F}$ is acyclic, then so is $psc$.

Proof.

1. We show by induction on $n$, that $[F^{sc}]; hb; eco'; ([RW^{sc}]; T; [RW^{sc}])^{n}; eco'; hb; [F^{sc}] \subseteq psc^{+}_{F}$ for every $n \geq 0$. For $n = 0$, the claim holds since $eco'; eco; hb; [F^{sc}] \subseteq psc_{F}$. Suppose now that $[F^{sc}]; hb; eco'; ([RW^{sc}]; T; [RW^{sc}])^{n}; eco'; hb; [F^{sc}] \subseteq psc^{+}_{F}$, and let $R = [F^{sc}]; hb; eco'; ([RW^{sc}]; T; [RW^{sc}])^{n}; eco'; hb; [F^{sc}]$. Expanding the definition of $T$ (keeping in mind that $rmw \subseteq eco$) we have $R \subseteq R_1 \cup R_2$, where:

$$R_1 = [F^{sc}]; hb; eco'; ([RW^{sc}]; T; [RW^{sc}])^{n-1}; [RW^{sc}]; (sb; \cup sb; \neq loc; hb; sb; \neq loc); [RW^{sc}]; eco'; hb; [F^{sc}],$$

$$R_2 = [F^{sc}]; hb; eco'; ([RW^{sc}]; T; [RW^{sc}])^{n-1}; eco'; hb; [F^{sc}] .$$

Since $eco; eco; hb; [F^{sc}]$, by the induction hypothesis, we have $R_2 \subseteq psc^{+}_{F}$. In addition, since $sb; \neq loc; hb; sb; \neq loc; sb', hb; sb'$, our assumption entails that $R_1$ is contained in

$$R_1^{'} = [F^{sc}]; hb; eco'; ([RW^{sc}]; T; [RW^{sc}])^{n-1}; [F^{sc}]; hb; eco'; hb; [F^{sc}],$$

which, in turn, using the induction hypothesis is contained in $psc^{+}_{F}$.

2. Contrapositively, suppose that $psc$ is cyclic. Then, by definition, the union of the following relations are cyclic:

- $A_1 = [RW^{sc}]; T; [RW^{sc}]$
- $A_2 = [F^{sc}]; hb; (sb \cup eco); hb; [F^{sc}]$
- $A_3 = [RW^{sc}]; (sb \cup eco); hb; [F^{sc}]$
- $A_4 = [F^{sc}]; hb; (sb \cup eco); [RW^{sc}]$

Consider first the case that $A_1$ is cyclic. Then, since $rmw \subseteq eco$, the relation $[RW^{sc}]; (sb; \cup sb; \neq loc; hb; sb; \neq loc); [RW^{sc}] \cup eco$ is cyclic. Our assumption on $G$ entails that $hb; [F^{sc}]$; $hb \cup eco$ is cyclic. Since both $hb; [F^{sc}]$; $hb$ and $eco$ are transitive and irreflexive, we obtain that $hb; [F^{sc}]$; $hb; eco$ is cyclic, which in turn implies that $[F^{sc}]; hb; eco; hb; [F^{sc}] \subseteq psc_{F}$ is cyclic.

Now, consider the case that $A_1$ is acyclic. Then, the union of the following two relations must be cyclic:

- $B_1 = [F^{sc}]; hb; (sb \cup eco); ([RW^{sc}]; T; [RW^{sc}])^{*}; (sb \cup eco); hb; [F^{sc}]$
- $B_2 = [F^{sc}]; hb; (sb \cup eco); [F^{sc}]$

Note that $B_2 \subseteq psc^{+}_{F}$. In addition, we have $[F^{sc}]; hb; (sb \cup eco) \subseteq [F^{sc}]; hb; eco; and (sb \cup eco); hb; [F^{sc}] \subseteq eco; hb; [F^{sc}]$. By item 1, it follows that $B_1 \subseteq psc^{+}_{F}$ as well, and so $psc^{+}_{F}$ is cyclic.

Remark E.1. The proof of Lemma E.1 can be easily adapted for the following alternative $psc$ relations:

$$psc_{0} \triangleq ([E^{sc}] \cup [F^{sc}]; sb; (sb \cup eco); ([E^{sc}] \cup sb; [F^{sc}])$$

$$psc_{2} \triangleq ([E^{sc}] \cup [F^{sc}]; sb; (sb \cup sb'; hb; sb \cup eco); ([E^{sc}] \cup sb; [F^{sc}])$$

$$psc_{3} \triangleq ([E^{sc}] \cup [F^{sc}]; hb; (sb \cup sb'; hb; sb \cup eco); ([E^{sc}] \cup hb; [F^{sc}])$$

This ensures the correctness of the transformation for all alternative fixes presented in §2.

Lemma E.2. Let $G$ be an $RC11$-consistent execution without any $SC$ accesses. Let $A \subseteq \mathbb{R}_{acc} \cup \mathbb{R}_{rel}$, such that $[A]; (sb' \cup sb'; hb; sb'); [A]; hb; [F^{sc}]; hb$, and $[A]; rmw = rmw; [A]$. Then, the execution $G'$ obtained from $G$ by changing all modes of events in $A$ to $sc$ is $RC11$-consistent.

Proof. The only constraint that is affected by such modification is $SC$. Now, in $G'$ we have $[G'; RW^{sc}]; (G'; sb' \cup G'; sb'); (G'; hb; G'; sb'); (G'; RW^{sc}) \subseteq G'; hb; (G'; F^{sc}); G'; hb$, and by Lemma E.1 it suffices to show that $G'.psc_{F}$ is acyclic. This follows from the fact that $G$ satisfies $SC$, since $G'.psc_{F} = G.psc_{F}$. □
F. Properties of the Power and ARMv7 Models

In this appendix we provide the full definition of preserved program order (ppo) used by Power and ARMv7, and prove various properties of these models that are needed in our compilation correctness proof.

F.1 Preserved Program Order

ppo is defined based on the four dependencies — data, addr, ctrl, ctrl\_isync — that satisfy the following properties:

1. data \( \subseteq R \times W \).
2. addr \( \subseteq R \times (R \cup W) \).
3. ctrl\_isync \( \subseteq ctrl \subseteq R \times E \).
4. ctrl\_isync; sb \( \subseteq ctrl \).
5. ctrl\_isync; sb \( \subseteq ctrl \).
6. rmw \( \subseteq data \cup addr \cup ctrl \).
7. rmw; sb \( \subseteq ctrl \).

1 – 5 hold by definition (see [3]). 6 – 7 hold due to the compilation scheme: it always places a dependency from the load to the store that form an RMW pair, and a branch after each (conditional) store in such pairs.

The relation \texttt{deps} includes all types of dependencies:

\[ \texttt{deps} \triangleq \texttt{data} \cup \texttt{addr} \cup \texttt{ctrl} \]

Herd’s definition of ppo is as follows:

\[
\begin{align*}
\text{rdw} & \triangleq \text{rb}; \text{rfe} \cap \text{sb} \\
\text{detour} & \triangleq \text{moe}; \text{rfe} \cap \text{sb} \\
\text{ii}_0 & \triangleq \text{addr} \cup \text{data} \cup \text{rdw} \cup \text{rfi} \\
\text{ic}_0 & \triangleq \emptyset \\
\text{ci}_0 & \triangleq \text{ctrl}_{\text{isync}} \cup \text{detour} \\
\text{cc}_{\text{Power}} & \triangleq \text{data} \cup \text{ctrl} \cup \text{addr}; \text{sb}; \text{sb}_{\text{loc}} \\
\text{cc}_{\text{ARMv7}} & \triangleq \text{data} \cup \text{ctrl} \cup \text{addr}; \text{sb}; \\
\text{ppo} & \triangleq [R]; \text{ii}; [R] \cup [R]; \text{ic}; [W]
\end{align*}
\]

where, ii, ic, ci, cc are inductively defined as follows:

\[
\begin{array}{cccc}
\text{ii}_0 & \text{ci} & \text{ic};\text{ci} & \text{ii};\text{ii} \\
\text{ii} & \text{ii} & \text{ii} & \text{ii} \\
\text{ic};\text{ii} & \text{cc} & \text{ic};\text{cc} & \text{ii};\text{ic} \\
\text{ic} & \text{ic} & \text{ic} & \text{ic} \\
\text{ic};\text{ii} & \text{cc};\text{ci} & \text{ic};\text{cc} & \text{cc};\text{cc} \\
\text{ci} & \text{ci} & \text{ci} & \text{ci} \\
\text{cc} & \text{cc} & \text{cc} & \text{cc}
\end{array}
\]

Note that \( \text{ci} \subseteq \text{ii} \subseteq \text{ic} \), as well as \( \text{ci} \subseteq \text{cc} \subseteq \text{ic} \).

Alternatively the relations \text{ii}, \text{ic}, \text{ci}, \text{cc} can be defined as follows:

\[
xy \triangleq \bigcup_{n \geq 1} x^n y^n 
\]

where:

- \( x, y, x^1, \ldots, x^n, y^1, \ldots, y^n \in \{i, c\} \).
- If \( x = c \) then \( x^1 = c \).
- For every \( 1 \leq i \leq n - 1 \), if \( y^i = c \) then \( x^{i+1} = c \).
- If \( y = i \) then \( y^n = i \).
Note that the only difference between Power and ARMv7 is in the definition of $cc_0$. Henceforth, we only assume ARMv7’s definition, which is weaker, so our proofs apply for both Power and ARMv7.

Next, we prove some useful properties of $ppo$. In all propositions below we assume some Power-consistent execution.

**Proposition F.1.** $ppo$ is transitive.

**Proof.** Immediately follows from the definition. □

**Proposition F.2.** $[\mathcal{W}]; psbloc \subseteq ii$.

**Proof.** Let $(a, b) \in [\mathcal{W}]; psbloc$ and let $x = loc(a)$. Then, by definition, $a \in \mathcal{W}_x$, $b \in R_x$, $(a, b) \in sb$, and there is no $c \in \mathcal{W}_x$ such that $(a, c), (c, b) \in sb$. Since $G$ is complete, there exists some $d \in \mathcal{W}_x$ such that $(d, b) \in rf$. If $d = a$, then we are done since $rfi \subseteq ii$. Otherwise, since $G$ satisfies SC-PER-LOC, we have $(a, d) \in mo, (d, a) \not\in sb$, and $(b, d) \not\in sb$. It follows that $(a, d) \in moe$ and $(b, d) \in rfe$. Thus, we have $(a, b) \in detour \subseteq ii$. □

**Proposition F.3.** $(\text{deps} \cup \text{addr}; sb)\mathcal{W}; psbloc; ppo; [\mathcal{W}] \subseteq ppo$.

**Proof.** Let $a, b, c, d \in E$ such that $(a, b) \in (\text{deps} \cup \text{addr}; sb); [\mathcal{W}], (b, c) \in psbloc$, and $(c, d) \in ppo; [\mathcal{W}]$. If $(a, b) \in ctrl$, then by definition, we have $(a, d) \in ctrl$, and so $(a, d) \in ppo$. If $(a, b) \in addr; sb$, then by definition, we have $(a, d) \in cc$, and so $(a, d) \in ppo$. Otherwise, $(a, b) \in addr \cup data \subseteq ii$. By Prop. F.2, we also have $(b, c) \in ii$. Hence, $(a, c) \in ii$, and so $(a, c) \in ppo$. It follows that $(a, d) \in ppo$. □

**Proposition F.4.** $(\text{deps} \cup \text{addr}; sb)\mathcal{W}; |R|; sb; [\mathcal{W}] \subseteq ppo$.

**Proof.** Let $a, b, c \in E$ such that $(a, b) \in (\text{deps} \cup \text{addr}; sb); [\mathcal{W}]$, and $(b, c) \in sb; [\mathcal{W}]$. If $(a, b) \in ctrl$, then by definition, we have $(a, c) \in ctrl$, and so $(a, c) \in ppo$. Otherwise, $(a, b) \in addr; sb^\bot$. In this case, we have $(a, c) \in addr; sb$, and so $(a, c) \in ppo$. □

**Proposition F.5.** Let $R = \text{deps} \cup \text{addr}; sb \cup psbloc$. Then, $(\text{deps} \cup \text{addr}; sb); R^x; [\mathcal{W}] \subseteq ppo$.

**Proof.** We prove by induction that for every $n \geq 0$, $(\text{deps} \cup \text{addr}; sb); R^n; [\mathcal{W}] \subseteq ppo$. For $n = 0$, we have $(\text{deps} \cup \text{addr}; sb); [\mathcal{W}] \subseteq ppo$ by definition. Let $n \geq 1$ and suppose that $(\text{deps} \cup \text{addr}; sb); R^k; [\mathcal{W}] \subseteq ppo$ for every $k < n$. Let $(a, b) \in (\text{deps} \cup \text{addr}; sb); R^n; [\mathcal{W}]$. Let $c \in E$ such that $(a, c) \in (\text{deps} \cup \text{addr}; sb)$, and $(c, b) \in R^n$. If $c \in R$, then we are done using Prop. F.4. Otherwise, $c \in \mathcal{W}$, and $(c, b) \in psbloc; R^{n-1}$. Let $d$ be the sb-maximal event satisfying $(c, d) \in psbloc$ and $(d, b) \in R^k$ for some $k \leq n - 1$. The maximality of $d$ ensures that $(d, b) \in (\text{deps} \cup \text{addr}; sb); R^{k-1}$. By the induction hypothesis, we have $(d, b) \in ppo$. Hence, we have $(a, b) \in (\text{deps} \cup \text{addr}; sb); [\mathcal{W}]; psbloc; ppo; [\mathcal{W}]$, and the claim follows by Prop. F.3. □

**Proposition F.6.** Let $R = \text{deps} \cup \text{addr}; sb \cup psbloc$. Then, $rfe; R^x; [\mathcal{W}] \subseteq rfe; ppo$.

**Proof.** Let $(a, c) \in rfe; R^x; [\mathcal{W}]$. Let $b$ be the sb-maximal event satisfying $(a, b) \in rfe$ and $(b, c) \in R^x$. If $(b, c) \in (\text{deps} \cup \text{addr}; sb); R^x$, then we are done by Prop. F.5. Otherwise, let $d$ be the sb-maximal element such that $(b, d) \in psbloc$ and $(d, c) \in R^x$. Then, $d \in E$, and since $c \in \mathcal{W}$, we have $(d, c) \in R^x$. The maximality of $b$ and SC-PER-LOC ensure that $(b, d) \in rdw$, and so $(b, d) \in ppo$. The maximality of $d$ ensures that $(d, c) \in (\text{deps} \cup \text{addr}; sb); R^x$. By Prop. F.5, we have $(d, c) \in ppo$, and so $(a, c) \in rfe; ppo$. □

**Proposition F.7.** $ppo^?; rbi \subseteq ppo; mo^? \cup mo \cup rbi$.

**Proof.** For any $n \geq 0$, let $ppo_n$ denote ppo-edges that are formed by at most $n$ basic ppo edges $(i_{c_0}, i_{c_0}, c_{i_0}$, and $cc_0)$. Then, $ppo^? = \bigcup_{n \geq 0} ppo_n$. The proof proceeds by induction on $n$. For $n = 0$, the claim obviously holds. Suppose now that it holds for $n - 1$, and let $(a, b) \in ppo_n$ and $(b, c) \in rbi$. Then, $b$ must be a read event, and so there exists $a'$ such that $(a, a') \in ppo_{n-1}$ and $(a', b) \in ii_0 \cup cc_0$. This leads to five cases:

- $(a', b) \in addr$. In this case we have $(a', c) \in cc_0$, and so $(a, c) \in ppo$.
- $(a', b) \in rdw$. In this case we have $(a', c) \in rbi$, and the claim follows by the induction hypothesis.
\[(a',b) \in rfi.\] In this case we have \(\langle a',c \rangle \in mo,\) and so \(\langle a,c \rangle \in ppo^2;mo.\)

\[(a',b) \in ctrl_{\text{sync}}.\] In this case we have \(\langle a',c \rangle \in c_{10},\) and so \(\langle a,c \rangle \in ppo.\)

\[(a',b) \in detour.\] In this case we have \(\langle a',c \rangle \in mo,\) and so \(\langle a,c \rangle \in ppo^2;mo.\)

\[\square\]

### F.2 Additional Properties

**Proposition F.8.** \(rmw \cap (rb;mo) = \emptyset.\)

*Proof.* POWER-ATOMICITY condition ensures that \(rmw \cap (rb;mo) = \emptyset.\) In addition, in every execution we have \(rmw \subseteq sb, rbe; sb \not\subseteq sb; mo; mo \not\subseteq sb,\) and \(sb \not\subseteq rmw.\) It follows that \(rmw \cap (rb;mo) = \emptyset.\) \[\square\]

**Proposition F.9.** Let \(R \in \{\text{sync}; \text{fence}\}.\) Then, \(R; hb^*; rb \subseteq R; hb^*; \text{mo}^2.\)

*Proof.* We prove by induction on \(n\) that for every \(n \geq 0,\) we have \(R; hb^*_n; rb \subseteq R; hb^*_n; \text{mo}^2.\) For \(n = 0,\) the claim follows since \(R; rb \subseteq R.\) Now, suppose it holds for \(n - 1,\) and let \(a, b, c, d\) such that \(\langle a, b \rangle \in R; hb^{n-1}_n, \langle b, c \rangle \in hb^*_n,\) and \(\langle c, d \rangle \in rb\). If \(\langle b, c \rangle \in rfe,\) then we have \(\langle b, d \rangle \in mo,\) and so \(\langle a, d \rangle \in R; hb^*_n; \text{mo}.\) If \(\langle b, c \rangle \in \text{fence},\) then we have \(\langle b, d \rangle \in \text{fence},\) and so \(\langle a, d \rangle \in R; hb^*_n.\) Otherwise, we have \(\langle b, c \rangle \in ppo,\) and the claim follows using Prop. F.7 and the induction hypothesis. \[\square\]

**Proposition F.10.** \(\text{fence is transitive.}\)

*Proof.* Immediately follows from the definition of \(\text{fence}.\) \[\square\]

**Proposition F.11.** \(\text{fence}; hb^* \subseteq sb \cup \text{fence}; [w]; hb^*.\)

*Proof.* Let \(a,b,c \in E\) such that \(\langle a,b \rangle \in \text{fence}\) and \(\langle b,c \rangle \in hb^*_n.\) If \(\langle b,c \rangle \in sb,\) then the claim follows since \(\text{fence} \subseteq sb.\) Suppose otherwise. Then, there exists \(\langle d,e \rangle \in rfe\) such that \(\langle b,d \rangle \in hb^*_n \cap sb^\prime\) and \(\langle e,c \rangle \in hb^*_n.\) It follows that \(\langle a,d \rangle \in \text{fence},\) and so \(\langle a,c \rangle \in \text{fence}; [w]; hb^*.\) \[\square\]

**Proposition F.12.** \([\text{rb}; sb; (\text{fence}; hb^*^2); \text{sync} \subseteq (\text{fence}; hb^*^2); \text{sync}.\]

*Proof.* Immediately follows from the definition of \(\text{sync}\) and Prop. F.11. \[\square\]

**Proposition F.13.** \(\text{eco}^2; (\text{fence}; hb^*_n)^2; \text{sync}; hb^*_n\) is acyclic.

*Proof.* By definition, we have \(\text{eco}^2 = (\text{mo} \cup \text{rbe})^2; \text{rfi}^2 \cup \text{rb}; \text{rfi}^2 \cup \text{rb}; \text{rfe}.\) Thus, it suffices to show that the union of the following relations is acyclic:

- \(A = ((\text{mo} \cup \text{rbe})^2; \text{rfi} \cup \text{rb}; \text{rfi}^2); (\text{fence}; hb^*_n)^2; \text{sync}; hb^*_n\)
- \(B = \text{rbi}; \text{rfe}; (\text{fence}; hb^*_n)^2; \text{sync}; hb^*_n\)

By Prop. F.9, \(A; B \subseteq A; A\) and \(B; B \subseteq B; A.\) Hence, it suffices to show that \(A\) is acyclic and \(B\) is irreflexive. Acyclicity of \(A\) follows from Power’s PROPAGATION condition, since we have \(A \subseteq \text{mo}; \text{prop}_2\) (using Prop. F.12). Irreflexivity of \(B\) also follows from PROPAGATION, using Prop. F.9. \[\square\]

**Proposition F.14.** Let \(A = \{a \in W \mid \exists b \in F. \langle b,a \rangle \in sb\}_\text{imm}; rmw^2.\)

Then, \((sb^2; [F]; sb \cup [A]; mo^2); \text{rfe}; hb^*; (sb; [F])^2)\) is a strict partial order.

*Proof.* Let \(R = (sb^2; [F]; sb \cup [A]; mo^2); \text{rfe}; hb^*; (sb; [F])^2.\) The fact that \(R\) is transitive follows from the following facts (obtained by expanding the relevant definitions):

- \(sb; [F]; sb \cup [A]; mo^2); \text{rfe}; hb^*; (sb; [F])^2.\) The fact that \(R\) is transitive follows from the following facts (obtained by expanding the relevant definitions):

  - \(sb; [F]; sb \cup [A]; mo^2); \text{rfe}; hb^*; (sb; [F])^2.\)
  - \(rfe; hb^*; (sb; [F]; sb \cup [A]; mo^2); \text{rfe}; hb^*; (sb; [F])^2.\)

Now, to see that \(R\) is irreflexive, note that \(\langle a,a \rangle \in R\) implies (using these three properties) that \(\langle a,a \rangle \in hb^*_n\) which contradicts POWER-NO-THIN-AIR. \[\square\]

**Proposition F.15.** \(\text{eco}; (sb \cup \text{fence}; hb^*_n)\) is irreflexive.
Proof. \textit{eco}; \textit{sb} is irreflexive using \textsc{sc-per-loc}. By Prop. F.11, it suffices to show that \textit{eco}; \textit{fence}; \textit{wb} ; \textit{hb} \textsuperscript{+} is irreflexive. Suppose otherwise, and let \(a, b \in E\) such that \(\langle a, b \rangle \in \textit{eco}\) and \(\langle b, a \rangle \in \textit{fence}; \textit{wb}\). First, if \(\langle a, b \rangle \in \textit{sb}\), then we have \(\langle a, a \rangle \in \textit{fence}; \textit{hb} \textsuperscript{+} \subseteq \textit{hb} \textsuperscript{+}\), which contradicts \textsc{power-no-thin-air}. Suppose otherwise, and consider the possible cases:

- \(\langle a, b \rangle \in \textit{rfe}\). In this case we obtain \(\langle a, a \rangle \in \textit{hb} \textsuperscript{+}\), which contradicts \textsc{power-no-thin-air}.
- \(\langle a, b \rangle \in \textit{mo}; \textit{rf} \textsuperscript{2}\). Let \(c \in E\) such that \(\langle a, c \rangle \in \textit{mo}\) and \(\langle c, b \rangle \in \textit{rf} \textsuperscript{2}\). Then, we have \(\langle c, a \rangle \in \textit{prop}\), and we obtain that \textit{mo}; \textit{prop} \textsubscript{1} is not irreflexive, which contradicts \textsc{propagation}.
- \(\langle a, b \rangle \in \textit{rbe}; \textit{rf} \textsuperscript{2}\). Let \(c \in W\) such that \(\langle a, c \rangle \in \textit{rbe}\) and \(\langle c, b \rangle \in \textit{rf} \textsuperscript{2}\). Let \(d \in W\) such that \(\langle b, d \rangle \in \textit{fence}\) and \(\langle d, a \rangle \in \textit{hb} \textsuperscript{+}\). Then, we have \(\langle c, d \rangle \in \textit{prop}\), and obtain a violation of \textsc{observation}.
- \(\langle a, b \rangle \in \textit{rbi}; \textit{rfe}\). Let \(c \in W\) such that \(\langle a, c \rangle \in \textit{rbi}\) and \(\langle c, b \rangle \in \textit{rfe}\). By Prop. F.9, we have \(\langle b, c \rangle \in \textit{fence}; \textit{hb} \textsuperscript{+}; \textit{mo} \textsuperscript{2}\). Let \(d \in E\) such that \(\langle b, d \rangle \in \textit{fence}; \textit{hb} \textsuperscript{+}\) and \(\langle d, c \rangle \in \textit{mo} \textsuperscript{2}\). Then, we have \(\langle c, d \rangle \in \textit{prop}\), and we obtain that \textit{mo} \textsuperscript{2}; \textit{prop} \textsubscript{1} is not irreflexive, which contradicts \textsc{propagation}. \qed

F.3 Removing Redundant Fences

Lemma F.1. Let \(G\) be a \textsc{power} execution, and let \(\langle a, b \rangle \in [\textit{f\langle sync\rangle}; \textit{sb}\textsubscript{\textit{imm}}; [\textit{f\langle sync\rangle}].\) Let \(G'\) be the execution obtained from \(G\) by removing \(b\) (\(G' = G_{\{(G, \exists, \langle b \rangle)\}}\)). If \(G'\) is \textsc{power}-consistent, then so is \(G\).

Proof. Since \(b\)'s immediate \textit{sb}-predecessor is a full fence, we have \(G'.\textit{fence} = G.\textit{fence}\). Then, it is easy to see that for every relation \(c\) mentioned in Def. 5, we have \(G'.c = G.c\), and so if \(G'\) is \textsc{power}-consistent, then so is \(G\). \qed

Lemma F.2. Let \(G\) be a \textsc{power} execution, and let \(\langle a, b \rangle \in [\textit{R}; \textit{sb}\textsubscript{\textit{imm}} \cap \textit{ctrl}_{\textit{sync}}]; \textit{F}\). Let \(G'\) be the execution obtained from \(G\) by removing the \textit{ctrl}_{\textit{sync}} dependency edges from \(a\) onwards (\(G'.\textit{ctrl}_{\textit{sync}} = G.\textit{ctrl}_{\textit{sync}} \setminus \{(a, E)\}\)). If \(G'\) is \textsc{power}-consistent, then so is \(G\).

Proof. Since \(a\)'s immediate \textit{sb}-successor is a fence, we have \(\langle a, c \rangle \in G.\textit{fence}\) for every \(c \in RW\) such that \(\langle a, c \rangle \in \textit{sb}\). Now, by omitting \textit{ctrl}_{\textit{sync}} dependency edges from \(a\) onwards, we may remove \textit{ppo}-edges from \(a\), but whenever \textit{ppo} is used to form an \textit{hb}-edge, it can be replaced by a \textit{fence}-edge. Consequently, for every relation \(c\) mentioned in Def. 5, we have \(G'.c = G.c\), and so if \(G'\) is \textsc{power}-consistent, then so is \(G\). \qed

G. Power-before Relation

In this section, we define a relation that we call \textit{power-before} (\textit{pb}), and show that if \textit{pb} is acyclic in some execution \(G\) of a program \(P\), then either \(G\) is RC11-consistent, or \(P\) has undefined behavior under RC11. This relation is the key for showing that \textsc{no-thin-air} holds when proving compilation correctness. (Thus, if one is only interested in weakRC11-consistency, this section can be completely ignored.)

In what follows we assume an execution \(G\).

\textit{pb} is given by:

\[
\begin{align*}
\textit{psbloc} & \equiv \textit{sb}\mid_{loc}; [\textit{R}] \setminus \textit{sb}\mid_{loc}; [\textit{W}] ; \textit{sb} & \quad \text{(preserved sb-loc)} \\
\textit{pb} & \equiv \textit{deps} \cup \textit{addr}; \textit{sb} \cup [\textit{R}_{\textit{rlx}} \cup \textit{W}_{\textit{rlx}} \cup \textit{F}]; \textit{sb} \cup \textit{psbloc} \cup \textit{sb}; [\textit{E}_{\textit{rel}}] & \quad \text{(internal Power-before)} \\
\textit{pb} & \equiv \textit{pb}\mid_{\textit{rf}} & \quad \text{(Power-before)}
\end{align*}
\]

Clearly, \(\textit{pb} \subseteq \textit{sb} \cup \textit{rf}\), and so \(\textit{pb}\) is acyclic in every RC11-consistent execution.

Proposition G.1. If \(G\) is weakRC11-consistent, then \(\textit{rf} \subseteq \textit{pb}\).

Proof. \textsc{coherence} guarantees that \(\textit{rfi} \subseteq \textit{psbloc} \subseteq \textit{pb}\), and by definition we have \(\textit{rfe} \subseteq \textit{pb}\). \qed

Proposition G.2. For every weakRC11-consistent execution \(G\), \(\textit{hb} \subseteq \textit{sb} \cup \textit{pb}\).

Proof. It suffices to show that \(\textit{sb}\mid_{\textit{rel}}; \textit{swe}; \textit{sb}\mid_{\textit{rel}} \subseteq \textit{pb}\). By definition, we have \(\textit{sb}\mid_{\textit{rel}}; \textit{swe}; \textit{sb}\mid_{\textit{rel}} \subseteq \textit{sb}\mid_{\textit{rel}}; [\textit{E}_{\textit{rel}}]; \textit{sb}\mid_{\textit{rel}}; (\textit{rf} \cup \textit{rmw})^{+}; [\textit{R}_{\textit{rlx}}]; \textit{sb}\mid_{\textit{rel}}\).
The claim follows because we have:

- \( \text{sb}^*; [\text{E}^{\text{rel}}]; \text{sb}^* \subseteq \text{pb}i^* \)
- \( \text{rf} \subseteq \text{pb} \) and \( \text{rmw} \subseteq \text{deps} \subseteq \text{pb}i. \)
- \( [\text{R}^{\text{rel}}]; \text{sb}^* \subseteq \text{pb}i^*. \)

\[ \Box \]

**Proposition G.3.** If \( \text{pb}i \) is acyclic, but \( \text{sb} \cup \text{rf} \) is cyclic, then \( (\text{rf}e; [\text{R}^0] \setminus \text{hb}); \text{sb} \neq \emptyset. \)

**Proof.** A cycle in \( \text{sb} \cup \text{rf} \) implies a cycle in \( \text{rfe}; \text{sb}. \) Since \( \text{rfe}; [\text{R}^{\text{rel}}]; \text{sb} \) and \( (\text{rfe} \cap \text{hb}); \text{sb} \) are contained in \( \text{pb}^+ \) (using Prop. G.2 for the latter), there must exist an edge \((a, b) \in \text{rfe}; \text{sb} \) that is neither in \( \text{rfe}; [\text{R}^{\text{rel}}]; \text{sb} \) nor in \((\text{rfe} \cap \text{hb}); \text{sb}. \) Then, we have \((a, b) \in (\text{rfe}; [\text{R}^0] \setminus \text{hb}); \text{sb}. \)

**Lemma G.1.** Suppose that \( G \) is a weakRC11-consistent execution of a program \( P, \) and that \( \text{pb}i \) is acyclic, but \( G \) is not RC11-consistent. Then, \( P \) has undefined behavior under RC11.

**Proof.** Since \( G \) is weakRC11-consistent but not RC11-consistent, we have that \( \text{sb} \cup \text{rf} \) is cyclic. By Prop. G.3, \( \text{rf}e; [\text{R}^0] \setminus \text{hb} \not\subseteq \text{hb}. \) We show that this implies that \( P \) has undefined behavior under RC11.

Let \( a_1, \ldots, a_n \) be an enumeration of \( \text{E} \) that respects \( \text{pb} \) (that is, \( i < j \) whenever \( (a_i, a_j) \in \text{pb}^+ \)). For every \( 1 \leq i \leq n, \) let \( E_i = \{a_1, \ldots, a_i\}. \) Let \( k \) be the minimal index such that \([E_k]; \text{rf}e; [\text{R}^0]; [E_k] \not\subseteq \text{hb}\). Then, we have \((a_j, a_k) \in \text{rf}e; [\text{R}^0] \setminus \text{hb} \) for some \( j < k. \) Let \( B = \text{dom}(\text{sb}^*; [E_k]) \) and \( H = B \setminus E_k. \)

**Claim 1:** \( h \in [\text{R}^0] \cup [\text{W}^{\text{rel}}] \) for every \( h \in H. \)

**Proof:** Otherwise, since \([\text{R}^{\text{rel}}] \cup [\text{W}^{\text{rel}}] \cup F; \text{sb} \subseteq \text{pb}, \) we would obtain \((h, a) \in \text{pb} \) for some \( a \in E_k. \) This contradicts the fact that \( h \notin E_k. \)

**Claim 2:** \( (h, b) \notin \text{sb}^* \) for every \( h \in H \) and \( b \in B \cap (\text{E}^{\text{rel}}). \)

**Proof:** Suppose otherwise. Let \( a \in E_k \) such that \((h, a) \in \text{sb}^* \). It follows that \((h, a) \in \text{sb}^*; \text{E}^{\text{rel}}; \text{sb}^*, \) and so \((h, a) \in \text{pb}^*. \) Hence, \( h \in E_k \) as well, which contradicts our assumption.

**Claim 3:** \( (h, b) \notin \text{deps}^*; \text{ctrl} \) for every \( h \in H \) and \( b \in B. \)

**Proof:** Suppose otherwise. Let \( a \in E_k \) such that \((h, a) \in \text{sb}^*. \) Since \( \text{ctrl}; \text{sb}^* \subseteq \text{ctrl}, \) it follows that \((h, a) \in \text{deps}^+, \) and so \((h, a) \in \text{pb}^+. \) This contradicts the fact that \( h \notin E_k. \)

**Claim 4:** \( (h, b) \notin \text{deps}^*; \text{addr} \) for every \( h \in H \) and \( b \in B. \)

**Proof:** Suppose otherwise. Let \( a \in E_k \) such that \((h, a) \in \text{sb}^*. \) Then, \((h, a) \in \text{deps}^*; \text{addr}; \text{sb}^* \subseteq \text{pb}^+. \) This contradicts the fact that \( h \notin E_k. \)

Let \( h_1, \ldots, h_m \) be an enumeration of \( H \) that respects \( \text{sb}, \) and let \( H_i = \{h_1, \ldots, h_i\} \) for every \( 0 \leq i \leq m. \)

**Claim 5:** For every \( 1 \leq i \leq m, \) \( h_i \notin \text{dom}(\text{deps}^*; [E_k \cup H_{i-1}]). \)

**Proof:** Suppose otherwise, and let \( a \in E_k \cup H_{i-1} \) such that \((h_i, a) \in \text{deps}^+. \) Then, \((h_i, a) \in \text{pb}^+. \) If \( a \in E_k, \) then \( h_i \in E_k \) as well, which contradicts our assumption. Hence, we have \((h_i, a) \in H_{i-1}. \) This contradicts the fact that the \( h_i \)’s enumeration respects \( \text{sb}. \)

**Claim 6:** Let \( 1 \leq i \leq m, \) and let \( x = \text{loc}(h_i). \) Let \( a \in (E_k \cup H_{i-1}) \cap \text{R}_x \) and suppose that \((h_i, a) \in \text{sb}. \) Then, \((h_i, a) \in \text{sb}; [(E_k \cup H_{i-1}) \cap \text{W}_x]; \text{sb}. \)

**Proof:** Suppose otherwise. Let \( 1 \leq j \leq m \) be the maximal index satisfying \( h_j \in E_x, \) \((h_i, h_j) \in \text{sb}^+ \) and \((h_i, a) \in \text{sb}. \) Then, \((h_j, a) \in \text{psloc}, \) and so \((h_j, a) \in \text{pb}. \) If \( a \in E_k, \) then \( h_j \in E_k \) as well, which contradicts our assumption. Hence, we have \((h_j, a) \in H_{i-1}. \) This contradicts the fact that the \( h_i \)’s enumeration respects \( \text{sb}. \)

For every \( 1 \leq i \leq n, \) let and \( G_i = G|_{E_i}. \) Since \( G.rfe \subseteq G.pb \) (Prop. G.1), all the \( G_i \)’s are weakRC11-consistent. Additionally, \( G_i.pb \) is acyclic for every \( 1 \leq i \leq n. \)

We inductively construct a sequences of labeling functions \( \text{lab}_0, \ldots, \text{lab}_m : B \rightarrow \text{Label} \) and executions \( G_0', \ldots, G_m' \) such that the following hold:

1. For every \( 0 \leq i \leq m, \) \( G_i'.E = E_k \cup H_i. \)
2. For every $0 \leq i \leq m$, $G'_i.1ab = lab_i|_{G'_i.E}$.
3. For every $0 \leq i \leq m$, $G'_i$ is RCna-consistent.
4. For every $0 \leq i \leq m$, $(aj, ak), (ak, aj) \notin G'_i.hb$.
5. For every $0 \leq i \leq m$, $lab_i(\langle G'_i.B \rangle)$ is an execution of $P$.
6. For every $0 \leq i \leq m$, $G'.rf^{-1}; G'_i.rf^{-1}; G'.rf; G.rmw \subseteq [G.E]$.

Finally, we would obtain that $G'm$ is a racy RCna-consistent execution with $G'm.E = B$, and $lab_m(\langle G'_m.B \rangle) = G'm.1ab(\langle G'_m.B \rangle)$ is an execution of $P$. Hence, $G'_m$ is an execution of $P$, and by Lemma D.1, $G'_m$ is RC11-consistent or $P$ has undefined behavior under RC11. Since $G'm$ is racy, in any case we would obtain that $P$ has undefined behavior under RC11.

First, we define $lab_0$ and $G'_0$. The minimality of $k$ and Prop. G.3 ensure that $G_{k-1}$ is RC11-consistent. Hence, Lemma D.4 ensures that there exists some event $b \in E_k$ such that the execution $G'$ given by $G'.c = G_k.c$ for every $c \in \{sb, rmw, data, addr, ctrl, mo\}$, $G'.lab = G_k.lab[a \mapsto R^a(G.loc(a_k), G.val_v(b))]$, and $G'.rf = G_k.rf \cup \{b, ak\}$ is RCna-consistent. In addition, $ak \notin dom(G'_B.deps)$ (since it is $G.pb$ maximal in $G|_B$). By Assumption B.1, there exists a reevaluation $lab$ of $G$.lab such that $lab_G|_B$ is an execution of $P$, $lab_G|_B.val_l(a_k) = G.val_v(b)$, and $lab(c) = G|_B.lab(c)$ for every $c \in B \setminus \{ak\}$. We take $lab_0 = lab$ and $G'_0 = G'$. It is straightforward to see that $lab$ and $G'$ satisfy the six conditions above. In particular, $G|_B.rmw^{-1}; G'.rf^{-1}; G'.rf; G|_B.rmw \subseteq [G.E]$ follows from the fact that $G$ satisfies ATOMICITY. Additionally, by Prop. C.4, $G'.hb = G_k.hb$, and so, we have $(aj, ak), (ak, aj) \notin G'.hb$.

Next, let $1 \leq i \leq m$, and suppose that $lab_{i-1}$ and $G'_{i-1}$ are defined. We construct $lab_i$ and $G'_i$. By Claim 1 above, we have $h_i \in G.R^{pb} \cup G.W^{rrix}. Let$ $G'_i be the execution obtained from $G'_{i-1}$ by adding the event $h_i$, labeled with $(lab_{i-1}(h_i))$, and the sb, rmw, and dependency edges from/to $h_i$ as in $G|_B$. By Claim 2 above, we also have $h_i \notin dom(G'_i, sb; [G'_i, E^{\text{ra}}])$. Let $x = G.loc(h_i)$, and consider the two cases:

$h_i \in G.R^{pb}$: Since $G'_{i-1}$ is RCna-consistent, Lemma D.4 ensures that there exists some event $b \in E_k \cup H_i$, $G'_{i-1}.lab = G'_{i-1}.lab \cup \{h_i \mapsto R^{pa}(x, G'_i.val_v(b))\}$, $G'.c = G'_i.c$ for every $c \in \{sb, rmw, data, addr, ctrl, mo\}$, and $G'.rf = G'_i.rf \cup \{b, ak\}$ is RCna-consistent. In addition, by Claims 3 and 4 above, we have that $h_i \notin dom(G'_B.deps); (G'_B.ctrl \cup G'_B.addr)$. By Assumption B.1, there exists a reevaluation $lab$ of $lab_{i-1}$ such that $lab_G|_B$ is an execution of $P$, $lab_G|_B.val_l(h_i) = G.val_v(b)$, and $lab(c) = lab_{i-1}(c)$ for every $c$ such that $(h_i, c) \notin G|_B.deps$. We take $lab_i = lab$ and $G'_i = G'$. Again, it is straightforward to see that $lab$ and $G'$ satisfy the required conditions. In particular, $G'_i.lab = lab_i|_{G'_i.E}$ follows from the fact that $G'_{i-1}.lab = lab_{i-1}|_{E_k \cup H_{i-1}}$, and Claim 5 above. In addition, by Prop. C.5, we have $[G'_{i-1}.E]; G'.hb; [G'_{i-1}.E] = G'_{i-1}.hb$, and so, we have $(aj, ak), (ak, aj) \notin G'.hb$.

$h_i \in G.W^{rrix}$: By Claim 6 above, we have that for every $b \in G'_i.E$, if $(h_i, b) \in G'_i, sb; [G'_i, R_x]$, then $(a, b) \in G'_i, sb; [G'_i, W_x]; G'_i, sb$. Thus, since $G'_{i-1}$ is RCna-consistent, and $G.rmw^{-1}; G'_{i-1}.rf^{-1}; G'{i-1}.rf; G.rmw \subseteq [G.E]$, Lemmas D.2 and D.3 ensure that there exists $T \subseteq G'_i.W_x \times G'_i.W_x$ such that the execution $G'$ given by $G'.E = E_k \cup H_i$, $G'.lab = lab_{i-1}|_{E_k \cup H_i}$, $G'.c = G'_i.c$ for every $c \in \{sb, rmw, data, addr, ctrl\}$, $G'.rf = G'_i.rf$, and $G'.mo = G'_i.mo \cup T$ is RCna-consistent. We take $lab_i = lab_{i-1}$ and $G'_i = G'$. It is straightforward to see that $lab_{i-1}$ and $G'$ satisfy the required conditions. In particular, Prop. C.3 guarantees that $(aj, ak), (ak, aj) \notin G'.hb$. \qed

H. Proof of Compilation Correctness

Lemma H.1. Let $G$ be an execution without SC accesses. Let $G_p$ be a Power-execution. Suppose that the following hold:

- $G.data \subseteq G_p.data; G.addr \subseteq G_p.addr$, and $G.ctrl \subseteq G_p.ctrl$.
- $G.rmw; G.sb \subseteq G_p.ctrl$.
- $G.F^{sc} \subseteq G_p.F^{\text{sync}}$ and $G.F^{sc} = G_p.F^{\text{sync}}$.
- $G.W^{rel} \subseteq A$ where $A = \{a \in G_p, W | \exists b \in G_p.F: (b, a) \in G_p.sb|_{\text{imm}}; G_p.rmw^2\}$. 

25
• \([G.R^{1x}\setminus G.At]; G.sb \subseteq G.pctrl\).
• \([G.R^{acq}]; G.sb \subseteq G.p.rm^\uparrow; G.p.ctrl_{1sync} \).

Then:

• \(G\) and \(G_p\) have the same outcome.
• If \(G_p\) is Power-consistent, then \(G\) is weakRC11-consistent and \(G.pb\) is acyclic.

Proof. The first claim easily follows from our definitions. Suppose that \(G_p\) is Power-consistent. Before proving the second claim, we present some properties relating \(G\) and \(G_p\).

1. \(G.swe; G.sb^\uparrow \subseteq (G.p.sb^\uparrow; [G.p.F]; G.p.rub \cup [A]; G.p.mo_i^\uparrow); G.p.rm_e; G.p.hb^*; (G.p.sbb; [G.p.F])^2\)
   (follows from the definition of \(swe\))

2. \(G.hb \subseteq G.p.sbb \cup (G.p.sb^\uparrow; [G.p.F]; G.p.sbb \cup ([A] \cup G.p.rm_w); G.p.mo_i^\uparrow); G.p.rm_e; G.p.hb^*; (G.p.sbb; [G.p.F])^2\)
   (follows from Item 1 using Prop. F.14; note that \(G.p.sbb; [A] \subseteq G.p.sb^\uparrow; [F]; G.p.sbb \cup G.p.rm_w\))

3. \([G.RW]; (G.sbb \setminus G.rm_w); G.hb^\uparrow \subseteq G.p.sbb \cup G.p.fence; G.p.hb^*; (G.p.sbb; [G.p.F])^2\)
   (again follows from Item 1 using Prop. F.14)

   (easily follows from Item 2)

In addition, in order to apply Prop. C.2 in the proof below, we note that:

• \([G.w]; G.sbb_{loc}; [G.w] \subseteq G.mo\): Indeed, we have \([G.w]; G.sbb_{loc}; [G.w] = [G.p.w]; G.p.sbb_{loc}; [G.p.w]\) and \(G.mo = G.p.mo\), and the claim follows by Power’s SC-\(PER-LOC\) condition.

• \(G.rm_w \subseteq G.rb\): Indeed, we have \(G.rm_w = G.p.rm_w\) and \(G.rb = G.p.rb\), and the claim follows by Power’s SC-\(PER-LOC\) and the fact that \(G\) is complete.

Next, we show that \(G\) is weakRC11-consistent. Clearly, it is complete (since \(G.R = G.p.R\) and \(G.rf = G.p.rf\)).

COHERENCE. We show that \(G.eco^\uparrow; G.hb\) is irreflexive. The irreflexivity of \(G.hb\) follows from Prop. F.14.

Now, applying Prop. C.2, it suffices to show that \(G.eco \cup G.eco; (G.sbb \setminus G.rm_w); G.hb^\uparrow \) is irreflexive. First, \(G.eco = G.p.eco\) is irreflexive because of SC-\(PER-LOC\). Second, by property 3 above, we have \(G.eco; (G.sbb \setminus G.rm_w); G.hb^\uparrow; [G.RW] \subseteq G.p.eco; (G.p.sbb \cup G.p.fence; G.p.hb^*)\). By Prop. F.15, \(G.p.eco; (G.p.sbb \cup G.p.fence; G.p.hb^*)\) is irreflexive.

ATOMICTY. By Prop. F.8, we have \(G.p.rm_w \cap (G.p.rb; G.p.mo) = \emptyset\). Then, \(G.rm_w \cap (G.rb; G.mo) = \emptyset\) immediately follows since \(G.rm_w = G.p.rm_w, G.rb = G.p.rb, and G.mo = G.p.mo\).

SC. We show that \(G.psc\) is acyclic. Assuming no SC accesses, we have \(G.psc = R_1 \cup R_2\) where \(R_1 = [G.F^{sc}]; G.hb; G.eco; G.hb; [G.F^{sc}]\) and \(R_2 = [G.F^{sc}]; G.hb; [G.F^{sc}]\). Since \(R_2\) is irreflexive and \(R_2^+; R_1 \subseteq R_1\), it suffices to prove the acyclicity of \(R_1\). To this end, we show that \(G.eco; G.hb; [G.F^{sc}]; G.hb; [G.RW]\) is acyclic. Applying Prop. C.2, it suffices to show that \(G.eco; (G.sbb \setminus G.rm_w); G.hb^\uparrow; [G.F^{sc}]; G.hb; [G.RW]\) is acyclic. Using properties 3-4 above (and applying several simple simplifications), it suffices to show that the following relation is acyclic:


Using the definition of \(sync\), this relation is equal to:

\[G.p.eco; (G.p.fence; G.p.hb^*)^2; G.p.sync; G.p.hb^*; [G.p.RW].\]

Its acyclicity then follows by Prop. F.13.

Next, we show that \(G.pb\) is acyclic. Suppose otherwise. Then, there are \(a_1, \ldots, a_n\) such that \(\langle a_i, a_{i+1} \rangle \in G.rf_e; G.pb^+\) for every \(1 \leq i \leq n\) (where \(a_{n+1} = a_1\)). We show that \(\langle a_i, a_{i+1} \rangle \in G.p.hb^+\) for every \(1 \leq i \leq n\) (which contradicts POWER-NO-THIN-AIR). Let \(1 \leq i \leq n\), and let \(b \in E\) such that \(\langle a_i, b \rangle \in G.rf_e = G.p.rf_e\) and \(\langle b, a_{i+1} \rangle \in G.pb^+.\)

If \(\langle b, a_{i+1} \rangle \in G.p.fence, then we are done since \(G.p.rf_e; G.p.fence \subseteq G.p.hb\). Otherwise, it follows that \(\langle b, a_{i+1} \rangle \in (G.p.deps \cup G.p.addr; G.p.sbb \cup G.p.peblunc)^+\). By Prop. F.6, we have \(\langle a_i, a_{i+1} \rangle \in G.p.rf_e; G.p.ppo \subseteq G.p.hb^+.\)
Lemma H.2. Given a program $P$ without SC accesses, every outcome of $\langle P \rangle$ under Power is an outcome of $P$ under RC11.

Proof. Given a full Power-consistent Power execution $G_p$ of $\langle P \rangle$, the compilation scheme (see Fig. 7) ensures that there exists some full execution $G$ of $P$ for which the properties of Lemma H.1 hold. Here we assumed that all RMW write attempts (stwcx) succeed in the first attempt. Indeed, otherwise, one could always remove the RMW reads (lwarx) that precede the failed stwcx attempts while preserving Power-consistency as well as the outcome of the execution. Now, Lemma H.1 ensures that $G$ has the same outcome as $G_p$, $G$ is weakRC11-consistent, and $G.pb$ is acyclic. By Lemma G.1, either $G.xb \cup G.Rf$ is acyclic (and NO-THIN-AIR holds) or $P$ has undefined behavior under RC11. In any case, we obtain that the outcome of $G_p$ is an outcome of $P$ under RC11. ☐

I. Proofs for §7 (Correctness of Program Transformations)

In this appendix, we state (and outline the proofs of) the properties that ensure the soundness of the transformations in §7. For this purpose, it is technically convenient to employ a different presentation of RMWs, that treat them as single events (like in C11). To this end, we consider RMW-executions, defined as the executions in §3, with the following exceptions:

- Labels in RMW-executions may also be $\text{RMW}^\text{src}(x, v_r, v_w)$ where $o \in \{r1x, acq, rel, acqrel, sc\}$. Both sets $G.R$ and $G.W$ include all events $a$ with $\text{typ}(a) = \text{RMW}$, while $G.RMW$ denotes the set of all events $a$ with $\text{typ}(a) = \text{RMW}$.
- RMW-executions do not include an rmw component.

RC11-consistency for RMW-executions is also defined as for executions, with the following exceptions:

- $G.rb \equiv \text{rf}^{-1}; \text{mo} \setminus [E]$.
- Instead of ATOMICITY we now require:
  $$\text{rf} \cap (\text{mo}; \text{mo}) = \emptyset.$$  

(ATOMICY-RMW)

The rest of the notions are defined for RMW-executions exactly as for executions above.

There exists a trivial one-to-one correspondence, denoted by $\sim$, between executions according to §3 and RMW-executions (the latter are obtained by collapsing rmw-edges to single RMW events).

Proposition I.1. Suppose that $G \sim G^{\text{src}}$ for some execution $G$ and RMW-execution $G^{\text{src}}$. Then:

- $G$ is RC11-consistent iff $G^{\text{src}}$ is RC11-consistent.
- $G$ is racy iff $G^{\text{src}}$ is racy.

Using this correspondence, we may define and prove the correctness of transformations on RMW-executions.

Lemma I.1 (Strengthening). Let $G_{\text{tgt}}$ be an RMW-execution, obtained from an RMW-execution $G_{\text{src}}$ by strengthening some access/fence modes ($G_{\text{src}.mod}(a) \subseteq G_{\text{tgt}.mod}(a)$ for every $a \in G_{\text{src}.E}$). Then:

- If $G_{\text{tgt}}$ is RC11-consistent, then so is $G_{\text{src}}$.
- If $G_{\text{tgt}}$ is racy, then so is $G_{\text{src}}$.

Proof. Easily follows from our definitions, because both properties are monotone with respect to the mode ordering. ☐

Lemma I.2 (Sequentialization). Let $G_{\text{tgt}}$ be an RMW-execution, and let $\langle a, b \rangle \in \text{sb} \setminus \text{sb}; \text{sb}$. Let $G_{\text{src}}$ be the RMW-execution obtained from $G$ by removing the sb-edge $\langle a, b \rangle$. Then:

- If $G_{\text{tgt}}$ is RC11-consistent, then so is $G_{\text{src}}$.
- If $G_{\text{tgt}}$ is racy, then so is $G_{\text{src}}$.

Proof. Easily follows from our definitions, because both properties are monotone with respect to sb. ☐

Next, to state the soundness of deordering transformations, we use the following definition of adjacency.
Definition I.1. Let $R$ be a strict partial order on a set $A$. A pair $(a, b) \in A \times A$ is called $R$-adjacent if the following hold for every $c \in A$:

- If $(c, a) \in R$ then $(c, b) \in R$.
- If $(b, c) \in R$ then $(a, c) \in R$.

Lemma I.3 (Non-load-store deordering). Let $G_{tgt}$ be an RMW-execution, and let $a, b \in G_{tgt}$. Such that $(a, b)$ is $G_{tgt}$.sb-adjacent. Let $G_{src}$ be the RMW-execution obtained from $G_{src}$ by adding an sb-edge $(a, b)$. Suppose that the labels of $a$ and $b$ form a deorderable pair according to Table 1, except for the load-store deorderable pairs $(R; W, R; RMW)$.

- If $G_{tgt}$ is RC11-consistent, then so is $G_{src}$.
- If $G_{tgt}$ is racy, then so is $G_{src}$.

Proof. It is straightforward to verify that all components and derived relations in $G_{src}$ are identical to those of $G_{tgt}$ except for: $G_{src}.sb = G_{tgt}.sb \cup \{\langle a, b \rangle\}$ and $G_{src}.hb = G_{tgt}.hb \cup \{\langle a, b \rangle\}$. Then, the fact that $G_{src}$ is RC11-consistent, easily follows from the fact that $G_{tgt}$ is RC11-consistent. In particular, since $a, b$ is not a load-store deorderable pair, assuming that $G_{tgt}$ satisfies NO-THIN-AIR, we cannot have $\langle b, a \rangle \in (G_{src}.sb \cup G_{src}.rf)^+$, so the additional sb-edge $(a, b)$ cannot close an sb $rb$ cycle. Finally, since $G_{src}.race = G_{tgt}.race$, we have that $G_{src}$ is racy if $G_{tgt}$ is racy.

Lemma I.4 (Load-store deordering). Let $G_{tgt}$ be an RMW-execution, and let $a, b \in G_{tgt}$. Such that $(a, b)$ is $G_{tgt}.sb$-adjacent. Let $G_{src}$ be the RMW-execution obtained from $G_{src}$ by adding an sb-edge $(a, b)$. Suppose that the labels of $a$ and $b$ form a load-store deorderable pair $(R; W, R; RMW)$, or $(R; W)$ according to Table 1. Then:

- If $G_{tgt}$ is RC11-consistent, then $G_{src}$ is weakRC11-consistent and $G_{src}.pb$ is acyclic.
- If $G_{tgt}$ is racy, then so is $G_{src}$.

Proof. The proof is similar to the proof of Lemma I.3. The fact that $G_{src}$ is weakRC11-consistent follows from the fact that $G_{tgt}$ is RC11-consistent. In addition, since $G_{src}.pb = G_{tgt}.pb \subseteq G_{src}.sb \cup G_{tgt}.rf$, assuming that $G_{tgt}$ satisfies NO-THIN-AIR, we have that $G_{src}.pb$ is acyclic.

Using Lemma G.1, one obtains the soundness of load-store deordering according to Table 1.

Notation I.1. For a binary relation $R$ on a set $A$ and an element $a \in A$, we denote by $R_+^a$ the set $\{b \in A | \langle b, a \rangle \in R\}$, and by $R_+^a$ the set $\{b \in A | \langle a, b \rangle \in R\}$.

Lemma I.5 (Read-read merging). Let $G_{tgt}$ be an RC11-consistent RMW-execution. Let $a \in R \setminus RMW$, and let $a' \in E$ such that $(a', a) \in rb$. Let $b \not\in E$, and let $G_{src}$ be the RMW-execution satisfying:

- $G_{src}.E = G_{tgt}.E \cup \{b\}$.
- $G_{src}.lab = G_{tgt}.lab \cup \{b \mapsto G_{tgt}.lab(a)\}$.
- $G_{src}.sb = G_{tgt}.sb \cup \{\langle a, b \rangle\} \cup (G_{tgt}.sb^+ \times \{b\}) \cup (\{b\} \times G_{tgt}.sb^+_a)$.
- $G_{src}.rf = G_{tgt}.rf \cup \{\langle a', b \rangle\}$.
- $G_{src}.mo = G_{tgt}.mo$.

Then, $G_{src}$ is RC11-consistent, and it is racy if $G_{tgt}$ is racy.

Proof. By definition, $G_{src}$ is complete, and ATOMICITY-RMW holds (since $G_{src}.mo = G_{tgt}.mo$ and $b \not\in G_{src}.R \setminus G_{src}.RMW$). It is also easy to see that we have:

- $G_{src}.eco = G_{tgt}.eco \cup (G_{tgt}.eco^+_a \times \{b\}) \cup (\{b\} \times G_{tgt}.eco^+_a)$.
- $G_{src}.hb = G_{tgt}.hb \cup \{\langle a, b \rangle\} \cup (G_{tgt}.hb^+_a \times \{b\}) \cup (\{b\} \times G_{tgt}.hb^+_a)$.

Hence, $G_{src}$ satisfies COHERENCE. To see that NO-THIN-AIR holds, note that if we had $\langle b, a \rangle \in (G_{src}.sb \cup G_{src}.rf)^+$, then we would have $\langle a, a \rangle \in (G_{tgt}.sb \cup G_{tgt}.rf)^+$, and, similarly, if we had $\langle b, a' \rangle \in (G_{src}.sb \cup G_{src}.rf)^+$, then we would have $\langle a, a' \rangle \in (G_{tgt}.sb \cup G_{tgt}.rf)^+$. It remains to show that $G_{src}.psc$ is acyclic. If $G_{tgt}.mod(a) \neq sc$, then we have $G_{src}.psc = G_{tgt}.psc$, and the claim follows since $G_{tgt}$ satisfies SC. Otherwise, we have:

- $G_{src}.psc = G_{tgt}.psc \cup \{\langle a, b \rangle\} \cup (G_{tgt}.psc^+_a \times \{b\}) \cup (\{b\} \times G_{tgt}.psc^+_a)$.
This implies that a $G_{src.psc} \cup G_{src.psc}$ cycle would imply a $G_{tgt.psc} \cup G_{tgt.psc}$ cycle. Finally, if $(c, b) \in G_{src.race}$, then we have $(c, a) \in G_{tgt.race}$. \hfill \Box

**Lemma I.6 (Write-write merging).** Let $G_{tgt}$ be an RC11-consistent RMW-execution. Let $b \in W \setminus \text{RMW}$, $a \not\in E$, and $v \in \text{Val}$. Let $G_{src}$ be the RMW-execution satisfying:

- $G_{src.E} = G_{tgt.E} \uplus \{a\}$.
- $G_{src.lab} = G_{tgt.lab} \uplus \{(a \mapsto \overline{w}^{G_{tgt.loc}(b)}(G_{tgt.loc}(b), v))\}$.
- $G_{src.sb} = G_{tgt.sb} \uplus \{(a, b)\} \cup (G_{tgt.sb}^+ \times \{a\}) \cup (\{a\} \times G_{tgt.sb}^+_a)$.
- $G_{src.rf} = G_{tgt.rf}$.
- $G_{src.mo} = G_{tgt.mo} \cup \{(a, b)\} \cup (G_{tgt.mo}^+ \times \{a\}) \cup (\{a\} \times G_{tgt.mo}^+_a)$.

Then, $G_{src}$ is RC11-consistent, and it is racy if $G_{tgt}$ is racy.

**Proof.** By definition, $G_{src}$ is complete. To see that ATOMICITY-RMW holds, note that we have $G_{src.mo; G_{src.mo}; \text{RMW}} \subseteq G_{tgt.mo; G_{tgt.mo} \cup \{(a) \times G_{src.E}\}}$, and that $a$ has no outgoing $\text{rf}$-edges. It is also easy to see that we have:

- $G_{src.eco} = G_{tgt.eco} \cup \{(a, b)\} \cup (G_{tgt.eco}^+ \times \{b\}) \cup (\{b\} \times G_{tgt.eco}^+_b)$.
- $G_{src.hb} = G_{tgt.hb} \cup \{(a, b)\} \cup (G_{tgt.hb}^+ \times \{b\}) \cup (\{b\} \times G_{tgt.hb}^+_b)$.

Hence, $G_{src}$ satisfies COHERENCE. To see that NO-THIN-AIR holds, note that if we had $(a, b) \in (G_{src.sb} \cup G_{src.rf})^+$, then we would have $(b, b) \in (G_{tgt.sb} \cup G_{tgt.rf})^+$, Since $G_{tgt}$ is racy, it remains to show that $G_{src.psc}$ is acyclic. If $G_{tgt.loc}(a) \neq \text{sc}$, then we have $G_{src.psc} = G_{tgt.psc}$, and the claim follows since $G_{tgt}$ satisfies SC. Otherwise, we have:

- $G_{src.psc} = G_{tgt.psc} \cup \{(a, b)\} \cup (G_{tgt.psc}^+ \times \{b\}) \cup (\{b\} \times G_{tgt.psc}^+_b)$.

This implies that a $G_{src.psc} \cup G_{src.psc}$ cycle would imply a $G_{tgt.psc} \cup G_{tgt.psc}$ cycle. Finally, if $(c, b) \in G_{src.race}$, then we have $(c, a) \in G_{tgt.race}$. \hfill \Box

**Lemma I.7 (Write/RMW-read merging).** Let $G_{tgt}$ be an RC11-consistent RMW-execution. Let $a \in W$ and $b \notin E$. Let $o \in \text{Ord}$, such that:

- If $\text{typ}(a) = \emptyset$ and $o = \text{sc}$, then $\text{mod}(a) = \text{sc}$.
- If $\text{typ}(a) = \text{RMW}$, then $a \subseteq \text{mod}(a)$.

Let $G_{src}$ be the RMW-execution satisfying:

- $G_{src.E} = G_{tgt.E} \uplus \{b\}$.
- $G_{src.lab} = G_{tgt.lab} \cup \{b \mapsto \overline{w}^{G_{tgt.loc}(a), G_{tgt.val}(a)}\}$.
- $G_{src.sb} = G_{tgt.sb} \cup \{(a, b)\} \cup (G_{tgt.sb}^+ \times \{b\}) \cup (\{b\} \times G_{tgt.sb}^+_b)$.
- $G_{src.rf} = G_{tgt.rf} \cup \{(a, b)\}$.
- $G_{src.mo} = G_{tgt.mo}$.

Then, $G_{src}$ is RC11-consistent, and it is racy if $G_{tgt}$ is racy.

**Proof.** Similar to the proof of Lemma I.5. \hfill \Box

**Lemma I.8 (Write-RMW merging).** Let $G_{tgt}$ be an RC11-consistent RMW-execution. Let $b \in W \setminus \text{RMW}$, $a \not\in E$, $v \in \text{Val}$, and $o \in \text{Ord}$ such that $\text{mod}(a) = \text{mod}(b)$. Let $G_{src}$ be the RMW-execution satisfying:

- $G_{src.E} = G_{tgt.E} \uplus \{a\}$.
- $G_{src.lab} = G_{tgt.lab} \cup \{b \mapsto \overline{w}^{G_{tgt.loc}(b), G_{tgt.val}(b)}\} \cup \{a \mapsto \overline{w}^{G_{tgt.loc}(b), v}\}$.
- $G_{src.sb} = G_{tgt.sb} \cup \{(a, b)\} \cup (G_{tgt.sb}^+ \times \{a\}) \cup (\{a\} \times G_{tgt.sb}^+_a)$.
- $G_{src.rf} = G_{tgt.rf} \cup \{(a, b)\}$.
- $G_{src.mo} = G_{tgt.mo} \cup \{(a, b)\} \cup (G_{tgt.mo}^+ \times \{a\}) \cup (\{a\} \times G_{tgt.mo}^+_a)$.

Then, $G_{src}$ is RC11-consistent, and it is racy if $G_{tgt}$ is racy.

**Proof.** By definition, $G_{src}$ is complete. To see that ATOMICITY-RMW holds, note that we have $G_{src.mo; G_{src.mo}; \text{RMW}} \subseteq G_{tgt.mo; G_{tgt.mo} \cup \{(a) \times G_{src.E}\}} \cup (G_{src.E} \times \{b\})$, and that $a$ has only an $\text{rf}$-edge to its immediate $G_{src.mo}$-successor $b$. The rest of the properties are proved as in the proof of Lemma I.6. \hfill \Box
Lemma I.9 (RMW-RMW merging). Let $G_{tgt}$ be an RC11-consistent RMW-execution. Let $a \in E$ with $\mathsf{lab}(a) = \mathsf{RMW}^o(x, v_r, v_w)$. Let $b \notin E$ and $v \in \mathsf{Val}$, and let $G_{src}$ be the RMW-execution satisfying:

- $G_{src}.E = G_{tgt}.E \cup \{b\}$.
- $G_{src}.\mathsf{lab} = G_{tgt}.\mathsf{lab}[a \mapsto \mathsf{RMW}^o(x, v_r, v)] \cup \{b \mapsto \mathsf{RMW}^o(x, v, v_w)\}$.
- $G_{src}.\mathsf{sb} = G_{tgt}.\mathsf{sb} \cup \{(a, b)\} \cup (G_{tgt}.\mathsf{sb}_a^+ \times \{b\}) \cup (\{b\} \times G_{tgt}.\mathsf{sb}_o^+)$.
- $G_{src}.\mathsf{rf} = G_{tgt}.\mathsf{rf} \cup \{(a, b)\}$.
- $G_{src}.\mathsf{mo} = G_{tgt}.\mathsf{mo} \cup \{(a, b)\} \cup (G_{tgt}.\mathsf{mo}_a^+ \times \{b\}) \cup (\{b\} \times G_{tgt}.\mathsf{mo}_o^+)$.

Then, $G_{src}$ is RC11-consistent, and it is racy if $G_{tgt}$ is racy.

Lemma I.10 (Fence-fence merging). Let $G_{tgt}$ be an RC11-consistent RMW-execution. Let $a \in F$, $b \notin E$, and let $G_{src}$ be the RMW-execution satisfying:

- $G_{src}.E = G_{tgt}.E \cup \{b\}$.
- $G_{src}.\mathsf{lab} = G_{tgt}.\mathsf{lab}[a \mapsto L \text{ where } L \text{ is some access label with mode } \mathsf{na} \text{ and location } x]$.
- $G_{src}.\mathsf{sb} = G_{tgt}.\mathsf{sb} \supset G_{tgt}.\mathsf{sb}$ and every event in $x$ is $G_{src}.\mathsf{sb}$-related to $a$.
- $G_{src}.\mathsf{rf} = G_{tgt}.\mathsf{rf}$ if $G_{src}.\mathsf{typ}(a) = \mathsf{W} \setminus \mathsf{RMW}$, and otherwise $G_{src}.\mathsf{rf} = G_{tgt}.\mathsf{rf} \cup \{\max_{G_{src}.\mathsf{sb}} G_{src}.\mathsf{sb}_o^+, a\}$.

Then, $G_{src}$ is RC11-consistent, and it is racy if $G_{tgt}$ is racy.

Proof. By definition, $G_{src}$ is complete, and $\mathsf{ATOMICITY-RMW}$ holds since $G_{src}.\mathsf{rf} = G_{tgt}.\mathsf{rf}$ and $G_{src}.\mathsf{mo} = G_{tgt}.\mathsf{mo}$. It is also easy to see that we have $G_{src}.\mathsf{eco} = G_{tgt}.\mathsf{eco}$ and:

- $G_{src}.\mathsf{hb} = G_{tgt}.\mathsf{hb} \cup \{(a, b)\} \cup (G_{tgt}.\mathsf{hb}_a^+ \times \{b\}) \cup (\{b\} \times G_{tgt}.\mathsf{hb}_o^+)$.

Hence, $G_{src}$ satisfies $\mathsf{COHERENCE}$. To see that $\mathsf{NO-THIN-AIR}$ holds, note that if we had $(a, a) \in G_{src}.\mathsf{sb} \cup G_{src}.\mathsf{rf}$, then we would have $(a, a) \in G_{tgt}.\mathsf{sb} \cup G_{tgt}.\mathsf{rf}$. It remains to show that $G_{src}.\mathsf{psc}$ is acyclic. If $G_{tgt}.\mathsf{mod}(a) \neq \mathsf{sc}$, then we have $G_{src}.\mathsf{psc} = G_{tgt}.\mathsf{psc}$, and the claim follows since $G_{tgt}$ satisfies $\mathsf{sc}$. Otherwise, we have:

- $G_{src}.\mathsf{psc} = G_{tgt}.\mathsf{psc} \cup \{(a, b)\} \cup (G_{tgt}.\mathsf{psc}_a^+ \times \{b\}) \cup (\{b\} \times G_{tgt}.\mathsf{psc}_o^+)$.

This implies that a $G_{src}.\mathsf{psc}$ cycle would imply a $G_{tgt}.\mathsf{psc}$ cycle. Finally, $G_{src}.\mathsf{race} = G_{tgt}.\mathsf{race}$, so $G_{src}$ is racy if $G_{tgt}$ is racy.

Soundness of register promotion is proved in two steps. First, we show that if all accesses to some location are in one thread, then they can be safely weakened to non-atomic accesses. Second, we show that these non-atomic accesses can be safely removed (replaced by register assignments at the program level).

Lemma I.11 (Register promotion-a). Let $G_{tgt}$ be an RC11-consistent RMW-execution. Suppose that all accesses to some location $x$ are related by $G_{tgt}.\mathsf{sb}$. Let $G_{src}$ be the RMW-execution obtained by strengthening the accesses mode of all accesses to $x$ to $\mathsf{sc}$. Then, $G_{src}$ is RC11-consistent, and it is racy if $G_{tgt}$ is racy.

Proof. By definition, we have $G_{src}.c = G_{tgt}.c$ for $c \in \{\mathsf{sb}, \mathsf{rf}, \mathsf{mo}, \mathsf{eco}\}$. It is also easy to see that $G_{src}.\mathsf{hb} = G_{tgt}.\mathsf{hb}$. Hence, $G_{src}$ is complete, and $\mathsf{ATOMICITY, COHERENCE, NO-THIN-AIR}$ hold for $G_{src}$ since they hold for $G_{tgt}$. To see that $G_{src}.\mathsf{psc}$ is acyclic, it suffices to note that $G_{src}.\mathsf{psc} \subseteq G_{tgt}.\mathsf{psc} \cup G_{tgt}.\mathsf{sb}$ (acyclicity of $G_{tgt}.\mathsf{psc}$ follows from the acyclicity of $G_{tgt}.\mathsf{psc}$ since $\mathsf{sb} \subseteq \mathsf{psc}$ in every execution). Finally, if $(a, b) \in G_{tgt}.\mathsf{race}$ and $\mathsf{na} \in \{G_{tgt}.\mathsf{mod}(a), G_{tgt}.\mathsf{mod}(b)\}$, then the same holds in $G_{src}$: we must have $\mathsf{loc}(a) \neq x$ if $(a, b) \notin G_{tgt}.\mathsf{hb} \cup (G_{tgt}.\mathsf{hb})^{-1}$.

Lemma I.12 (Register promotion-b). Let $G_{tgt}$ be an RC11-consistent RMW-execution. Let $x \in \mathsf{Loc}$ and let $X = \{b \in E \mid \mathsf{loc}(b) = x\}$. Suppose that all accesses in $X$ are related by $G_{tgt}.\mathsf{sb}$. Let $a \notin E$, let $G_{src}$ be an RMW-execution satisfying:

- $G_{src}.E = G_{tgt}.E \cup \{a\}$.
- $G_{src}.\mathsf{lab} = G_{tgt}.\mathsf{lab} \cup \{a \mapsto L\}$ where $L$ is some access label with mode $\mathsf{na}$ and location $x$.
- $G_{src}.\mathsf{sb} \supset G_{tgt}.\mathsf{sb}$ and every event in $x$ is $G_{src}.\mathsf{sb}$-related to $a$.
- $G_{src}.\mathsf{rf} = G_{tgt}.\mathsf{rf}$ if $G_{src}.\mathsf{typ}(a) = \mathsf{W} \setminus \mathsf{RMW}$, and otherwise $G_{src}.\mathsf{rf} = G_{tgt}.\mathsf{rf} \cup \{\max_{G_{src}.\mathsf{sb}} G_{src}.\mathsf{sb}_o^+, a\}$.
Then, \(G_{\text{src}}\) is RC11-consistent, and it is racy if \(G_{\text{tgt}}\) is racy.

\(\square\)

### J. Proofs for §8 (Programming Guarantees)

**Theorem 3.** If in all SC-consistent executions of a program \(P\), every race \(\langle a, b \rangle\) has \(\text{mod}(a) = \text{mod}(b) = \text{sc}\), then the outcomes of \(P\) under RC11 coincide with those under SC.

**Proof.** Let \(P\) be a program, and suppose that every race \(\langle a, b \rangle\) in some SC-consistent execution of \(P\) has \(\text{mod}(a) = \text{mod}(b) = \text{sc}\). We prove that \(P\) has no weak behaviors. Suppose toward a contradiction that there exists an execution \(G\) of \(P\) that is RC11-consistent but not SC-consistent. (Note that if \(P\) has undefined behavior under RC11, then there exists a racy RC11-consistent execution of \(P\), and our assumption ensures that this execution is not SC-consistent.)

We call an execution \(G'\) a prefix of an execution \(G\) if it is obtained by restricting \(G\) to a set \(E\) of events that contains the set \(E_0\) of initialization events, and is closed with respect to \(G_{\text{sb}} \cup G_{\text{rf}}\) (\(a \in E\) whenever \(b \in E\) and \(\langle a, b \rangle \in G_{\text{sb}} \cup G_{\text{rf}}\)). It is easy to show that \(G'\) is RC11-consistent, provided that \(G\) is RC11-consistent.

**Notation J.1.** For an execution \(G\), \(G_{\text{rf}|_{\text{sc}}}\) denotes the restriction of \(G_{\text{rf}}\) to SC accesses \((G_{\text{rf}}|_{\text{sc}} = [G_{\text{E}}']; G_{\text{rf}}; [G_{\text{E}}^\ast'])\). A similar notation is used for \(G_{\text{mo}}\) and \(G_{\text{rb}}\).

For a set of events \(E\), let \(\Pi(E)\) denote the set of all pairs \(\langle a, b \rangle \in E \times E\) of conflicting events, such that \(\{G_{\text{mod}}(a), G_{\text{mod}}(b)\} \neq \{\text{sc}\}\) and \(\langle a, b \rangle, \langle b, a \rangle \notin (G_{\text{sb}} \cup G_{\text{rf}}|_{\text{sc}})^+\). Let \(a_1, \ldots, a_n\) be an enumeration of \(E \setminus E_0\) that respects \(G_{\text{sb}} \cup G_{\text{rf}}\) (that is, \(i < j\) whenever \(\langle a_i, a_j \rangle \in G_{\text{sb}} \cup G_{\text{rf}}\)). For every \(1 \leq i \leq n\), let \(E_i = E_0 \cup \{a_1, \ldots, a_i\}\) and \(G_i = G|_{E_i}\). Since the \(G_i\)'s are all prefixes of \(G\), all of them are RC11-consistent.

**Claim:** For every \(1 \leq i \leq n\), if \(\Pi(E_i) = \emptyset\) then \(G_i\) is SC-consistent.

**Proof:** Suppose that \(\Pi(E_i) = \emptyset\). Since \(G\) satisfies COHERENCE, it follows that:

- \(G_i_{\text{rf}} \subseteq (G_{\text{sb}} \cup G_{\text{rf}}|_{\text{sc}})^+\).
- \(G_i_{\text{mo}} \subseteq (G_{\text{sb}} \cup G_{\text{rf}}|_{\text{sc}})^+ \cup G_{\text{mo}}|_{\text{sc}}\).
- \(G_i_{\text{rb}} \subseteq (G_{\text{sb}} \cup G_{\text{rf}}|_{\text{sc}})^+ \cup G_{\text{rb}}|_{\text{sc}}\).

Hence, we have \(G_i_{\text{sb}} \cup G_i_{\text{rf}} \cup G_i_{\text{mo}} \cup G_i_{\text{rb}} \subseteq R^+\), where \(R = G_{\text{sb}} \cup G_{\text{rf}}|_{\text{sc}} \cup G_{\text{mo}}|_{\text{sc}} \cup G_{\text{rb}}|_{\text{sc}}\). Since \(G\) satisfies the SC condition, we have that \(R\) is acyclic, and so \(G_i\) is SC-consistent (ATOMICITY holds since it holds for \(G\)).

Now, since \(G\) is not SC-consistent, we have \(\Pi(G|_{E}) \neq \emptyset\). Let \(k = \min\{i \mid \Pi(E_i) \neq \emptyset\}\). Then, \(\Pi(E_{k-1}) = \emptyset\) (and so, \(G_{k-1}\) is SC-consistent), and there exists some \(j < k\), such that \(a_j\) and \(a_k\) are conflicting, \(\{G_{\text{mod}}(a_j), G_{\text{mod}}(a_k)\} \neq \{\text{sc}\}\), and \(\langle a_j, a_k \rangle, \langle a_k, a_j \rangle \notin (G_{\text{sb}} \cup G_{\text{rf}}|_{\text{sc}})^+\). Let \(B = \{b \in E_k \mid \langle b, a_j \rangle \in G_{\text{sb}}\}\). Since \(\langle a_j, a_k \rangle \notin (G_{\text{sb}} \cup G_{\text{rf}}|_{\text{sc}})^+\), and \(G_{k-1,\text{rf}} \subseteq (G_{\text{sb}} \cup G_{\text{rf}}|_{\text{sc}})^+\), we have \(\langle a_j, b \rangle \notin (G_{\text{sb}} \cup G_{\text{rf}})^+\) for every event \(b \in B\). Let \(x = \text{loc}(a_k)\), and consider two cases:

- \(\text{typ}(a_k) = W:\)
  - **Claim:** \(\langle a_j, a_k \rangle \in G_{k,\text{race}}\).
    - **Proof:** Clearly, we have \(\langle a_k, a_j \rangle \notin (G_{k,\text{sb}} \cup G_{k,\text{rf}})^+\) (\(a_k\) has no outgoing sb and rf edges in \(G_k\)). In addition, we have \(\langle a_j, a_k \rangle \notin (G_{k,\text{sb}} \cup G_{k,\text{rf}})^+\) (otherwise, \(\langle a_j, b \rangle \in (G_{k,\text{sb}} \cup G_{k,\text{rf}})^+\) for some \(b \in B\)).

- **Claim:** \(G_k\) is not SC-consistent.
  - **Proof:** Since \(\langle a_j, a_k \rangle \in G_{k,\text{race}}\) and \(G_{\text{mod}}(a_j), G_{\text{mod}}(a_k)\) \(\neq \{\text{sc}\}\), the claim follows from our assumption.

- **Claim:** \(a_k \notin G_{\text{At}}\).
Proof: Suppose otherwise, and let $b \in G.E$ such that $\langle b, a_k \rangle \in \text{rmw}$. Since $G_k$ is not SC-consistent, but $G_{k-1}$ is SC-consistent, it must be the case that $\langle a_k, c \rangle \in G.m$ and $\langle c, a_k \rangle \in (G.sb \cup G.\text{rf} \cup G.mo \cup G.rb)^+$ for some $c \in E_{k-1}$. Let $d \in E_{k-1}$ such that $\langle c, d \rangle \in (G.sb \cup G.\text{rf} \cup G.mo \cup G.rb)^*$ and $\langle d, a_k \rangle \in G.sb \cup G.mo \cup G.rb$. Then, we also have $\langle c, d \rangle \in (G_{k-1}.sb \cup G_{k-1}.\text{rf} \cup G_{k-1}.mo \cup G_{k-1}.rb)^+$. If $\langle d, a_k \rangle \in G.mo \cup G.rb$, then we obtain $\langle d, c \rangle \in G.mo \cup G.rb$, and so $\langle d, c \rangle \in G_{k-1}.mo \cup G_{k-1}.rb$, which contradicts the fact that $G_{k-1}$ is SC-consistent. Otherwise, $\langle d, a_k \rangle \in G.sb$. It follows that $\langle d, b \rangle \in G.sb$. Now, COHERENCE ensures that $G.\text{rmw} \subseteq G.\text{rb}$, and it follows that $\langle b, c \rangle \in G.\text{rb}$. Hence, $\langle b, c \rangle \in G_{k-1}.rb$, which again contradicts the fact that $G_{k-1}$ is SC-consistent. 

Let $G'_k$ be the extension of $G_{k-1}$ with the event $a_k$ (with the same label as in $G_k$), the sb-edges of $G_k$, and the mo-edges $\{\langle a, b \rangle \mid a \in G_{k-1}.\text{\textunderscore}x\}$. It is easy to see that $G'_k$ is SC-consistent as well (in particular, it is important here that $a_k \not\in G.\text{at}$). Except for mo, it is identical to $G_k$, and so it is an execution of $P$ and $\langle a_j, a_k \rangle \in G'_k.\text{race}$. Since $\{G.\text{mod}(a_j), G.\text{mod}(a_k)\} \neq \{\text{sc}\}$, this contradicts our assumption.

- $\text{typ}(a_k) = R$:
  In this case, we must have $\text{typ}(a_j) = W$. Let
  \[ E = \{a \in G.E \mid \langle a, a_k \rangle \in (G.sb \cup G_{k-1}.\text{rf})^* \lor \langle a, a_j \rangle \in (G.sb \cup G_{k-1}.\text{rf})^*\}. \]
  Let $G'$ be the restriction of $G_k$ to the events in $E$. Since $G'|_{E \setminus \{a_k\}}$ is a prefix of $G_{k-1}$, it is SC-consistent. Let $c = \max_{G.\text{so}} G'.\text{\textunderscore}x$, and consider two cases.
  
  - $c \neq a_j$:
    Let $G''$ be the execution obtained from $G'$ by (i) modifying the value read at $a_k$ to $\text{val}_a(c)$, and (ii) adding the reads-from edge $\langle c, a \rangle$. It is easy to see that $G''$ is SC-consistent, and Assumption B.1 ensures that it is an execution of $P$. Additionally, $\langle a_j, a_k \rangle \notin (G''.sb \cup G''.\text{rf})^+$ (there are no outgoing sb and rf edges from $a_j$ in $G''$), and so, $\langle a_j, a_k \rangle \in G''.\text{race}$. Since $\{G.\text{mod}(a_j), G.\text{mod}(a_k)\} \neq \{\text{sc}\}$, this contradicts our assumption.
  
  - $c = a_j$:
    Let $d$ be the immediate $G.\text{mo}$-predecessor of $c$, and let $G''$ be the execution obtained from $G'$ by (i) modifying the value read at $a_k$ to $\text{val}_a(d)$, and (ii) adding the reads-from edge $\langle d, a \rangle$. Again, it is easy to see that $G''$ is SC-consistent, and Assumption B.1 ensures that it is an execution of $P$. As in the previous case we obtain a contradiction to our assumption. 

\[ \square \]

Theorem 4. Let $G$ be an RC11-consistent execution. Suppose that for every two distinct shared locations $x$ and $y$, $[E_x]; sb; [E_y] \subseteq sb; [F^{sc}]; sb$. Then, $G$ is SC-consistent.

Proof. It suffices to show that $sb \cup \text{ecoe}$ is acyclic (recall that $\text{ecoe} = \text{eco} \setminus \text{sb}$). Consider a cycle in $sb \cup \text{ecoe}$ of a minimal length. Cycles with at most one $\text{ecoe}$ edge are ruled out by COHERENCE. Hence, our cycle must have at least two $\text{ecoe}$ edges. Let $a_1, b_1, a_2, b_2, \ldots, a_n, b_n \in E$ (where $n \geq 2$) such that $\langle a_i, b_i \rangle \in \text{ecoe}$ and $\langle b_i, a_{i+1} \rangle \in sb$ for every $1 \leq i \leq n$ (where we take $a_{n+1} = a_1$). The events $a_1, b_1, \ldots, a_n, b_n$ are all accesses to shared locations (since $\langle a_i, b_i \rangle \in \text{ecoe} \subseteq \text{lwc}$ for every $1 \leq i \leq n$). In addition, we have $\text{lloc}(b_i) \neq \text{lloc}(a_{i+1})$ for every $1 \leq i \leq n$ (otherwise we would have $\langle a_i, a_{i+1} \rangle \in \text{ecoe}; sb|_{\text{lwc}} \subseteq \text{ecoe}$, which contradicts the minimality of the cycle). Therefore, our assumption entails that there exist $f_1, \ldots, f_n \in F^{sc}$ such that $\langle b_i, f_i \rangle \in sb$ and $\langle f_i, a_{i+1} \rangle \in sb$ for every $1 \leq i \leq n$. It follows that $\langle f_i, f_{i+1} \rangle \in F^{sc}; sb; \text{ecoe}; sb; [F^{sc}] \subseteq \text{psc}$ for every $1 \leq i \leq n$ (where we take $f_{n+1}$ to be $f_1$). This contradicts the fact that $G$ satisfies the SC constraint. 

\[ \square \]