Studying Operational Models of Relaxed Concurrency *

Gustavo Petri

Purdue University

Abstract. We study two operational semantics for relaxed memory models. Our first formalization is based on the notion of write-buffers which is pervasive in the memory models literature. We instantiate the (Total Store Ordering) TSO and (Partial Store Ordering) PSO memory models in this framework. Memory models that support more aggressive relaxations (eg. read-to-read reordering) are not easily described with write-buffers. Our second framework is based on a general notion of speculative computation. In particular we allow the prediction of function arguments, and execution ahead of time (eg. by branch prediction). While technically more involved than write-buffers, this model is more expressive and can encode all the Sparc family of memory models: TSO, PSO and (Relaxed Memory Ordering) RMO. We validate the adequacy of our instantiations of TSO and PSO by formally comparing their write-buffer and speculative formalizations. The use of operational semantics techniques is paramount for the tractability of these proofs.

1 Introduction

Current trends in multi-core architectures have raised interest in the formalization of relaxed memory models. While most works on the area concentrate on axiomatic definitions of such models [1,18,13] in this work we concentrate on the operational formalization of such models and the techniques that they enable.

Some recent works – including ours – have addressed the operational semantics of relaxed memory models [7,8,10,19,4], to mention but a few. In [7] we consider the operational semantics of *write-buffering* (see [1]). In these models writes to the memory are delayed in buffers, and are later updated into the memory. When a write is buffered, the issuing process is able to continue executing provided that its execution does not conflict with the suspended writes. Buffering a thread write has, from another thread perspective, the effect of delaying its execution w.r.t. subsequent actions of that thread. Write buffering is only one of the many memory order relaxations of common machine architectures and programming languages [1]. In [8] we consider the semantics of *speculative computation*, where actions in a thread can be performed in advance – or in parallel – without waiting for prior actions to be completed. Speculations are more general than write buffering, since more behaviors are possible. This additional

* Research supported by NSF 1237923.

$$\begin{bmatrix} p := 1\,; \\ r_0 := (!q) \end{bmatrix} \parallel \begin{bmatrix} q := 1\,; \\ r_1 := (!p) \end{bmatrix} \qquad \begin{bmatrix} p := 1\,; \\ q := 1 \end{bmatrix} \parallel \begin{bmatrix} r_0 := (!q)\,; \\ r_1 := (!p) \end{bmatrix}$$

(a) TSO & PSO: $r_0 = r_1 = 0$ (b) PSO: $r_0 = 1$ & $r_1 = 0$

Fig. 1: TSO and PSO examples

expressivity comes at the cost of a more elaborate formalism. Here we provide a uniform presentation and a formal comparison of instances of the frameworks justifying that claim. To do so, we present instances of both frameworks describing the TSO (Total Store Ordering) and PSO (Partial Store Ordering) memory models of Sparc [20]. However, the RMO (Relaxed Memory Ordering) model cannot be encoded with write-buffers.

Let us focus on TSO and PSO. Both models can reorder a read instruction with respect to a previous write. Figure 1a illustrates this behavior, where we assume that p and q are pointers in the memory initialized to 0, r_0 and r_1 refer to local "registers" (private to a thread), and we use the ML syntax $(!p)$ for dereferencing p. If we execute these threads according to their interleaving semantics [11] the final result $r_0 = r_1 = 0$ is not possible. However, if any of the reads is allowed to execute before its previous write (since they are on different references), the result is possible. PSO additionally allows two subsequent writes to be reordered. The result in Figure 1b can happen if the write of q takes place before the one of p. Another relaxation of these models is the capability of a thread to *read its own writes early*, according to [1]. Thus a thread can see its own writes before any other thread in the system.

In this work we formalize TSO and PSO with write buffers and speculations. For completeness we present a formalization of RMO with speculations. We then prove the adequacy of our formalizations of TSO and PSO with write-buffers and speculation. To that end we develop a third calculus, including both, write buffers and speculations, and prove their equivalence. This proof is based on the same basic concepts of true concurrency that we use to define the frameworks [7,8], where we distinguish as particularly important, the equivalence by permutation of independent steps, first introduced for the λ-calculus in [6].

In summary we make the following contributions: (1) We present in a uniform language two frameworks to describe relaxed memory models extending the ideas of [7,8]. (2) We present a speculative semantics that allows for *argument speculation*. While this is not our first attempt at speculative semantics [8], this addition generalizes the calculus presented in that work. In particular, branch prediction can be considered as a particular case of argument speculation where the condition is the argument. (3) As an example of the frameworks we instantiate the memory models of Sparc [20], which inspired some of the changes to [8]. (4) Using standard true concurrency techniques we prove the adequacy of the instantiations with the two frameworks of PSO and TSO. (5) This equivalence proof enables the reuse of the proof of the *fundamental property* of memory models, which we proved in [8] for buffered models, in the context of the speculative calculus. While the simulation argument of [8] is not surprising, a similar

argument in the calculus of speculations would require nonstandard techniques which is leveraged using our adequacy proof.

2 Two Frameworks of Relaxed Memory

To avoid clutter and focus on the memory model related aspects of programming languages we consider the syntax of a simple imperative call-by-value λ-calculus, extended with constructs for atomic operations and barriers to impose ordering among actions as typically found in the instruction set of machine architectures. We remark that the choice of a λ-like language constraints in no way the memory model arguments that follow.

$$\begin{array}{rcll} v \in \mathcal{V}al & ::= & x \mid \lambda x e \mid tt \mid ff \mid () & \textit{values} \\ e \in \mathcal{E}xpr & ::= & v \mid (ve) \mid (\mathsf{ref}\, v) \mid (\mathbf{!}\, v) \mid (v_0 := v_1) & \textit{expressions} \\ & \mid & (\mathsf{cas}\ v) \ \mid \ \langle \mathtt{wr}|\mathtt{rd}\rangle \ \mid \ \langle \mathtt{wr}|\mathtt{wr}\rangle & \end{array}$$

The memory model relevant instructions are writes to the memory, reads from the memory, atomic actions that use the memory and ordering instructions, all of them present in our language. Moreover we consider the language in quasi-Administrative Normal Form (ANF) [9][1].

Let us briefly discuss the intuitive semantics of the language. Our values are: λ-abstractions, booleans and the value $()$ to represent termination. We adopt the syntax $e_0\,; e_1$ to denote the expression $(\lambda x e_1 e_0)$ where x is not free in e_1. The expression $(\mathsf{ref}\, v)$ allocates a new memory location with the value v returning the reference where the value was allocated. The expression $(\mathbf{!}p)$ reads the memory retrieving the value at the location p. The expression $(p := v)$ updates the memory at location p with v. We also have a simple compare-and-swap construct $(\mathsf{cas}\ p)$ that atomically reads and modifies the reference p. In fact, this is a very primitive version of a standard read-modify-write construct. It reads the reference p and if the result of the read is ff it updates the location with value tt atomically; if the result of the read is tt it leaves the location unmodified. The returned value signals to the success or failure of the test. One could think of $(\mathsf{cas}\ p)$ as executing the following code atomically: $(\mathsf{if}\ (\mathbf{!}p)\ \mathsf{then}\ ff\ \mathsf{else}\ (p := tt)\,; tt)$. To finish with the language we have the barrier constructs $\langle \mathtt{wr}|\mathtt{rd}\rangle$ and $\langle \mathtt{wr}|\mathtt{wr}\rangle$ which are used to impose ordering on the evaluation of instructions of threads. These barrier instructions will not be of interest until the introduction of the relaxed semantics. We anticipate that the barrier $\langle \mathtt{wr}|\mathtt{rd}\rangle$ prevents write actions previous to the barrier (in the program syntax) from being delayed past read actions following the barrier. And similarly, the $\langle \mathtt{wr}|\mathtt{wr}\rangle$ barrier imposes constraints on two write instructions.

We present the technical tools that we will use throughout the paper alongside the standard semantics of this programming language. The operational semantics is given in two steps. First we provide rules that allow individual expressions to execute, where the values obtained from dereferencing a pointer are

[1] A more complete language is considered in [15].

$$\mathbf{E}[(\lambda x e v)] \xrightarrow{\beta_v} \mathbf{E}[(\lambda v^?\{x/v\}e\ v)] \qquad \mathbf{E}[(p := v)] \xrightarrow{\mathsf{wr}_{p,v}} \mathbf{E}[()] \qquad \mathbf{E}[(!\,p)] \xrightarrow{\mathsf{rd}^{\circ}_{p,v}} \mathbf{E}[v]$$

$$\mathbf{E}[(\lambda v^? e v)] \xrightarrow{\beta} \mathbf{E}[e] \qquad \mathbf{E}[(!\,p)] \xrightarrow{\mathsf{rd}_{p,v}} \mathbf{E}[v] \qquad \mathbf{E}[\langle \mathtt{wr}|\mathtt{rd}\rangle] \xrightarrow{\mathtt{wr}} \mathbf{E}[()]$$

$$\mathbf{E}[(\mathsf{ref}\ v)] \xrightarrow{\nu_{p,v}} \mathbf{E}[p] \qquad \mathbf{E}[(\mathsf{cas}\ p)] \xrightarrow{\mathsf{cas}_{p,v}} \mathbf{E}[v] \qquad \mathbf{E}[\langle \mathtt{wr}|\mathtt{wr}\rangle] \xrightarrow{\mathtt{ww}} \mathbf{E}[()]$$

Fig. 2: Semantics of Single Expressions

predicted (i.e. unconstrained). In a second step, we compose all the expressions (threads) into a single configuration that synchronizes them and interacts with the memory.

As it is common practice, we decompose expressions into a *redex* (reducible expression) and an evaluation context, that is an expression where a subexpression has been replaced with a hole denoted here by []. To describe the dynamics of our language we need to include pointers $p, q \in \mathcal{R}ef$ which are runtime values, and the runtime expression $(\lambda v^? e_0 e_1)$ which we use to decompose the β-reduction rule of the λ-calculus in two steps: $(\lambda x e_0 v) \xrightarrow{\beta_v} (\lambda v^?\{x/v\}e_0 v) \xrightarrow{\beta} \{x/v\}e_0$, where we include labels that will shortly be explained. The extended language is as follows:

$$\begin{array}{lcll}
e & ::= & \ldots \mid (\lambda v^? e_0\ e_1) & \textit{expressions} \\
v & ::= & \ldots \mid p \mid (\lambda v^? e_0) & \textit{values} \\
r & ::= & (\lambda x e v) \mid (\lambda v^? e v) \mid (\mathsf{ref}\ v) \mid (!\,p) \mid (p := v) & \textit{redexes} \\
 & \mid & (\mathsf{cas}\ p) \mid \langle \mathtt{wr}|\mathtt{rd}\rangle \mid \langle \mathtt{wr}|\mathtt{wr}\rangle & \\
\mathbf{E} & ::= & [] \mid (v\mathbf{E}) & \textit{evaluation contexts}
\end{array}$$

To describe the interaction of several threads and the memory, we label the transitions with the *actions* being taken at each step. Actions are sampled from the syntax:

$$\begin{array}{l}
a \in \mathcal{A}ct ::= \beta_v \mid \beta \mid \nu_{p,v} \mid \mathsf{wr}_{p,v} \mid \mathsf{rd}_{p,v} \mid \mathsf{rd}^{\circ}_{p,v} \mid \mathsf{cas}_{p,v} \mid b \\
b \in \mathcal{B}ar ::= \mathtt{wr} \mid \mathtt{ww}
\end{array}$$

The meaning of these symbols is better understood by looking at the semantics of single expressions in Figure 2. As we anticipated, β_v and β are the actions that result from a function application. Notice how the standard β-reduction rule is split into two steps. The actions concerning the memory are: $\nu_{p,v}$, which results from creating a new pointer p with value v, $\mathsf{wr}_{p,v}$ for writing on p, and similarly for $\mathsf{rd}_{p,v}$ where the value v is unconstrained. The action $\mathsf{cas}_{p,v}$ results from a compare-and-swap and $\mathtt{wr}$ and $\mathtt{ww}$ result from write-read barriers and a write-write barriers respectively. The special action $\mathsf{rd}^{\circ}_{p,v}$ notifies, in the relaxed semantics that follows, that the value v has been obtained from a buffer, or speculated.

The semantics of thread systems is given by means of transitions between configurations $C = (S, T)$ containing a *store* S, which represents the memory, and is formally a mapping from the set $\mathsf{dom}(S) \subseteq \mathcal{R}ef$ into values, and a thread

$$\frac{e \xrightarrow{a} e'}{(S, e_t \| T) \xrightarrow{a} (S', e'_t \| T)}\ (*) \begin{cases} a = \nu_{p,v} & \Rightarrow p \notin \mathsf{dom}(S)\ \&\ S' = S \cup \{p \mapsto v\} \\ a \in \{\mathsf{rd}_{p,v}, \mathsf{rd}^{\circ}_{p,v}\} & \Rightarrow S(p) = v \\ a = \mathsf{wr}_{p,v} & \Rightarrow S' = S[p \leftarrow v] \\ a \in \{\mathsf{cas}_{p,tt}\} & \Rightarrow S(p) = \mathit{ff}\ \&\ S' = S[p \leftarrow tt] \\ a \in \{\mathsf{cas}_{p,\mathit{ff}}\} & \Rightarrow S(p) = tt \end{cases}$$

Fig. 3: Multithreaded Semantics (Interleaving, or *Strong*)

system T which is a set of elements e_t where $t \in \mathcal{T}id$ is a thread identifier and $e \in \mathcal{E}xpr$ is the actual code of the thread. Of course, a thread identifier occurs at most once in T. We denote by $(e_t \| T)$ the thread system that contains all the threads in T as well as the thread e_t.

The semantics of the full thread system is given in Figure 3, where we only make explicit in the constraint $(*)$ the cases where store changes (i.e. $S' \neq S$), or where the action depends on S. We will consider this to be the *standard* semantics of our language. We will call this semantics the *strong* semantics, as opposed to the ones of the following sections, which we shall call *relaxed*, or *weak*.

Write Buffering Models. To formalize the semantics of TSO and PSO we add *write buffers* to the strong semantics. Buffers are FIFO queues of pending memory updates and barriers, defined by the syntax:

$$B ::= \ \epsilon \ \mid \ B \triangleleft [p \mapsto v] \ \mid \ B \triangleleft [b]$$

The empty buffer is denoted by ϵ, and nonempty buffers contain pending memory updates $[p \leftarrow v]$, or pending barriers $[b]$. We will use the notation $B(p)$ to denote the (ordered) sequence of values pending in the buffer B for the reference p, as well as any pending barriers on that buffer. The auxiliary function $B \downarrow p$ represents the buffer B where the first (in FIFO order) update to the reference p has been popped. We will also use the notation $a \triangleright B$ to represent the buffer whose first element is a and then continues like B.

We augment the thread systems of the previous section with buffers. In particular, since we are only concerned with the PSO and TSO memory models of Sparc there is no need to consider thread creation. The buffers are local to a thread, meaning that pending updates on buffers cannot be shared among different threads. Thus, configurations have now the form $C = (S, (B_t, e_t) \| T)$ where B_t is the buffer associated to the thread t. We will use new actions that result from updates pertaining buffers.

$$a \in \mathcal{A}ct \ ::= \ \ldots \ \mid \ \mathsf{bu}_{p,v} \ \mid \ \bar{b}$$

The action $\mathsf{bu}_{p,v}$ corresponds to an update of the memory by a write that was pending in a buffer, and the action $\bar{b}$ corresponds to the removal of a barrier $b \in \mathcal{B}ar$ from a buffer. We refer to these actions as the commit of a previous write or barrier that originated the buffer item. We can now present the semantics

$$\frac{e \xrightarrow{a} e'}{(S,(B,e_t)\|T) \xrightarrow[t]{a} (S',(B',e'_t)\|T)}\ (*) \qquad \frac{B = b \triangleright B' \qquad b \in \mathcal{B}ar}{(S,(B,e_t)\|T) \xrightarrow[t]{\bar{b}} (S,(B',e_t)\|T)}$$

$$\frac{B = [p \mapsto v] \triangleright B' \quad S' = S[p \leftarrow v]}{(S,(B,e_t)\|T) \xrightarrow[t]{\mathsf{bu}_{p,v}} (S',(B',e_t)\|T)}\text{TSO} \qquad \frac{B(p) = \mathtt{wr}^n \cdot v \cdot s \quad S' = S[p \leftarrow v]}{(S,(B,e_t)\|T) \xrightarrow[t]{\mathsf{bu}_{p,v}} (S',(B{\downarrow}p,e_t)\|T)}\text{PSO}$$

$$(*)\begin{cases} a = \nu_{p,v} \Rightarrow p \notin \mathsf{dom}(S)\ \&\ S' = S \cup \{p \mapsto v\} & \\ a = \mathsf{wr}_{p,v} \Rightarrow B' = B \triangleleft [p \mapsto v] & \\ a = \mathsf{rd}_{p,v} \Rightarrow S(p) = v\ \&\ \ulcorner B(p) \urcorner = \epsilon & \\ a = \mathsf{rd}^{\circ}_{p,v} \Rightarrow \ulcorner B(p) \urcorner = v & \end{cases} \qquad \begin{array}{l} a \in \mathcal{B}ar \Rightarrow B' = B \triangleleft [a] \\ a = \mathsf{cas}_{p,ff} \Rightarrow S(p) = tt\ \&\ B = \epsilon \\ a = \mathsf{cas}_{p,tt} \Rightarrow S(p) = ff\ \&\ B = \epsilon \\ \qquad\qquad \&\ S' = S[p \leftarrow v] \end{array}$$

Fig. 4: PSO & TSO with Write Buffers

of PSO and TSO in Figure 4. For convenience, we use the following notation on sequences[2] of pending writes and barriers: $\ulcorner s \urcorner = \epsilon$ if $s = \mathtt{ww}^n$; and $\ulcorner s \urcorner = v$ if $s = s' \cdot v \cdot \mathtt{ww}^n$; and $\mathtt{wr}$ does not occur in s', being undefined otherwise.

There are many rules that change from the strong semantics presented previously. Importantly, write and barrier actions have as their only effect appending the update or barrier to the end of the buffer. Notice as well that the rules for actions $\mathsf{rd}_{p,v}$ and $\mathsf{rd}^{\circ}_{p,v}$ are now different from each other. On the one hand, the action $\mathsf{rd}_{p,v}$ reads the contents from the memory, requiring that the buffer be empty for the reference p, and moreover, that there are no pending $\mathtt{wr}$ barriers. In fact, it is in this way that barrier symbols constrain the execution of some actions, disallowing particular reorderings. On the other hand, the action $\mathsf{rd}^{\circ}_{p,v}$ retrieves its value from the buffer, or reads its *own write*. The three new rules (i.e. the new actions) update the contents of the memory by emptying the buffers. These rules are nondeterministically triggered and model the asynchronous working of the memory architecture. In the rule for $\bar{b}$, the barrier symbol is removed from the buffer only when it reaches its top, that is, when all actions that were buffered previous to the barrier have been committed. Similarly, for TSO, a buffered write is updated into the memory upon reaching the top of the buffer. The only modification necessary to obtain PSO is the rule that updates the memory (that is the action $\mathsf{bu}_{p,v}$), where a buffered write can be committed into the store even if there are previously buffered writes on *different* references. These are the final two rules in Figure 4.

Let us reconsider the example program of Figure 1a. The following is a possible computation that justifies the result $r_0 = r_1 = 0$, where we demark the buffers with $\langle\rangle$, and implicitly name the thread to the left t_0 and the one to the

[2] Throughout the paper we use the notations $a \cdot b$ for the concatenation of sequences a and b, and $\leq$ for the prefix ordering.

right t_1 (we leave the PSO example Figure 1b to the reader):

$$\langle\epsilon\rangle(p := 1\,;\,(!q))\|\langle\epsilon\rangle(q := 1\,;\,(!p)) \xrightarrow[t_0]{\mathsf{wr}_{p,1}} \langle\epsilon \lhd [p \mapsto 1]\rangle(!q)\|\langle\epsilon\rangle(q := 1\,;\,(!p)) \xrightarrow[t_1]{\mathsf{wr}_{q,1}}$$
$$\langle\epsilon \lhd [p \mapsto 1]\rangle(!q)\|\langle\epsilon \lhd [q \mapsto 1]\rangle(!p) \xrightarrow[t_0]{\mathsf{rd}_{q,0}} \langle\epsilon \lhd [p \mapsto 1]\rangle(0)\|\langle\epsilon \lhd [q \mapsto 1]\rangle(!p)$$
$$\xrightarrow[t_1]{\mathsf{rd}_{p,0}}\xrightarrow[t_0]{\mathsf{bu}_{p,1}} \langle\epsilon\rangle 0 \parallel \langle\epsilon \lhd [q \mapsto 1]\rangle 0 \xrightarrow[t_1]{\mathsf{bu}_{q,1}} \langle\epsilon\rangle 0 \parallel \langle\epsilon\rangle 0$$

Importantly, both these memory models satisfy the *fundamental property* of relaxed memory models [2,17]. This property states that programs that are free of data races in their interleaving semantics only exhibit sequentially consistent behaviors in their relaxed semantics. We now make these claims precise.

Definition 1 (Data-Race). *A configuration C is said to contain a data race if $C = \big(S, (\mathbf{E}[(p := v)]_t \parallel \mathbf{E}'[r]_{t'} \parallel T')\big)$ and $r \in \{(!p), (p := w) \mid w \in \mathcal{V}al\}$.*

The definition of data-race can be easily lifted to programs.

Definition 2 (DRF Program). *We say a configuration C is data-race free (DRF for short) if every configuration C' reachable from C by the interleaving semantics (i.e. $C \xrightarrow{*} C'$) contains no data-race. A parallel program $e_0\| \ldots \|e_n$ is data-race free if the configuration $(\emptyset, e_0\| \ldots \|e_n)$ is data-race free.*

We can now prove the fundamental property for the models with write buffers.

Theorem 3 (Fundamental Property). *The weak memory models TSO and PSO implement the interleaving semantics for data-race free programs. More precisely, the configurations where all buffers are empty (c.f. Figure 4) reachable from a DRF regular configuration C in the semantics with write buffers coincide with the configurations reachable from the same configuration C in the interleaving semantics (c.f. Figure 3).*

We eschew the proof since it very closely follows the one in [7] with the simplification that here we do not consider thread creation. Moreover, since we do not consider locks (we are concerned with architectural models), the only means to establish ordering between concurrent conflicting accesses is through the use of cas instructions. Although the DRF result of TSO and PSO is well known, we emphasize here that our proof is a mostly standard bisimulation, which we can do due to the operational semantics. Most other proofs of this result are non-constructive (e.g. [13]).

As we anticipated, write buffers alone are not sufficient to model the read-read reorderings exhibited by RMO. This is typically illustrated by the IRIW (Independent Reads Independent Writes) example that follows, where we assume that p and q are initially 0:

$$\begin{bmatrix} r_0 := (!p)\,; \\ r_1 := (!q) \end{bmatrix} \parallel \begin{bmatrix} r_2 := (!q)\,; \\ r_3 := (!p) \end{bmatrix} \parallel p := 1 \parallel q := 1$$
$$\text{RMO: } r_0 = r_2 = 1 \;\&\; r_1 = r_3 = 0$$

It is clear that no write buffer behavior can produce this result since the writing threads have just one write each, and the reading threads do not use their write buffers in any way. It is therefore necessary to consider another kind of relaxation to capture RMO behaviors. If we allow any of the reading threads to speculate their second read before the first one, the behavior becomes possible (without recourse to any buffering argument). To see this, imagine that the second reads of the reading threads are executed first, then the writes of the writing threads, and finally the first reads of the reading threads. This kind of behavior is typical of *speculative* execution models which motivates our next semantics.

Speculative Models. We now consider the specification of relaxed memory models by means of speculations. We previously addressed the speculative semantics of programming languages in [8,15] where we generally discussed about the modeling of relaxed memory models. Let us briefly introduce a modified framework from [8] and show how TSO, PSO and RMO[3] can be modeled with it.

The two ingredients introduced for the speculative framework are *speculation contexts* and the *prediction of arguments* in applications. Speculation contexts generalize the evaluation contexts previously defined, by allowing the reduction of expressions that are otherwise not enabled in the strong semantics of Section 2, and have the following syntax:

$$\mathbf{\Sigma} ::= [] \mid (v\mathbf{\Sigma}) \mid (\lambda x\mathbf{\Sigma}e) \mid (\lambda v^{?}\mathbf{\Sigma}e) \qquad \textit{speculation contexts}$$

Notice that in an expression like $(\lambda x(\mathbf{!}p)e)$ one can reduce the redex $(\mathbf{!}p)$, by choosing the speculative context $(\lambda x[]e)$. In particular, this means that one can execute e_1 before e_0 in $(e_0\,;e_1)$. The second ingredient, the prediction of arguments, not present in [8], requires to extend the syntax for redexes with the new redex $(\lambda x e_0 e_1) \in r$, where the expression e_1 is not required to be a value. The speculative semantics for this redex is then:

$$\mathbf{\Sigma}[(\lambda x e_0 e_1)] \xrightarrow{\beta_v} \mathbf{\Sigma}[(\lambda v^{?}\ \{x/v\}e_0\ e_1)] \tag{1}$$

The reason why we need this type of speculation can be seen with the expression $(\lambda y\ p := y\ (q := 1\,;0))$. If we consider this example under the semantics of PSO with write buffers (cf. the previous section), it is clear that the write of p can be updated into the memory before the one of q, since they are on different references. However, if we are not able to predict that the expression $(q := 1\,;0)$ invariably returns 0, the write of p cannot proceed until the one of q has been performed with speculations. To solve this issue we add the possibility to predict the arguments (the reduction labeled β_v), which are later validated in the actual application (the reduction labeled β). Then, in the example above we can have:

$$\begin{aligned}(\lambda y\ p := y\ (q := 1\,;0)) &\xrightarrow{\beta_0} (\lambda 0^{?}\ p := 0\ (q := 1\,;0)) \\ &\xrightarrow{\mathsf{wr}_{p,0}} (\lambda 0^{?}()(q := 1\,;0)) \xrightarrow{*} (\lambda 0^{?}()0) \xrightarrow{\beta} ()\end{aligned}$$

[3] Since most proofs in this paper are not concerned with RMO we will just present its formalization for completeness, but we will otherwise ignore it.

$$\frac{e \xrightarrow[o]{a} e'}{(S, e_t \| T) \xrightarrow[t,o]{a} (S', e'_t \| T)}(*) \begin{cases} a \in \{\beta_v, \mathsf{rd}^{\circ}_{p,v}\} \Rightarrow \mathsf{FRef}(v) \subseteq \mathsf{dom}(S) \\ \cdots \end{cases}$$

Fig. 5: Speculative semantics

modeling the reordering of write buffers. Notice that if an argument is mispredicted, the speculation gets stuck, and therefore the computation is disregarded.

In fact, not all speculations will be considered legitimate. To define which ones we will regard as valid, we need to identify in the transitions *where* in the expression the reduction is taking place. To that end, we shall use *occurrences*, defined as sequences of symbols sampled from the set $\mathcal{SO}cc = \mathcal{O}cc \cup \{(\lambda\_[]\_)\}$ where $\mathcal{O}cc = \{(\_[])\}$. An occurrence $o \in \mathcal{O}cc^*$ is called *normal* in contrast with occurrences in the set $\mathcal{SO}cc^*/\mathcal{O}cc^*$ which we shall name *speculative*. Normal computations – that is computations that do not speculate – involve only normal occurrences. We can recover the occurrence $@\mathbf{\Sigma}$ of the hole in a speculation context $\mathbf{\Sigma}$ by means of the following inductive definition, where z in the last case can be a variable in $\mathcal{V}ar$ or a tagged value $v^?$:

$$@[] = \varepsilon \qquad @(v\mathbf{\Sigma}) = (\_[]) \cdot @\mathbf{\Sigma} \qquad @(\lambda z \mathbf{\Sigma} e) = (\lambda\_[]\_) \cdot @\mathbf{\Sigma}$$

We will denote by $e@o$ the subexpression of e whose occurrence is o in case that is defined. The inductive definition is obvious.

The semantics of expressions is similar to the one given in Figure 2 with the evaluation contexts $\mathbf{E}$ replaced by a speculation contexts $\mathbf{\Sigma}$ and the occurrence label ($@\mathbf{\Sigma}$) in the transitions. For example, the rule for β-reduction becomes: $\mathbf{\Sigma}[(\lambda v^? e v)] \xrightarrow[@\mathbf{\Sigma}]{\beta} \mathbf{\Sigma}[e]$. Only one rule is added, the one we presented in (1) for the redex $(\lambda x e_0 e_1)$ where e_1 is potentially not a proper value. In the example below one can see that these rules effectively achieve computing in advance w.r.t. the strong semantics of Section 2.

$$(!q)\,;p := tt \quad \xrightarrow{\mathsf{wr}_{p,tt}} \quad (!q)\,;() \quad \xrightarrow{\mathsf{rd}_{q,tt}} \quad tt\,;() \quad \xrightarrow{*} \quad ()$$

We can see that the write of p is performed before the read of q although the program text has them in the reverse order.

The semantics of thread systems is almost identical to the strong semantics presented in Figure 3. In fact the configurations are exactly the same, and the only rule that changes is the one for $\mathsf{rd}^{\circ}_{p,v}$, where the value is speculated at this stage of the semantics. The intention is that these values will be served by *own* writes of the same thread (cf. write buffers). The necessary conditions on this action will be imposed in the definition of valid computation. The semantics of thread systems, given in Figure 5, is almost identical to that of Figure 3 with the exception of the rules explicitly mentioned, where $\mathsf{FRef}(e)$ is the set of references occurring in e. The transitions are labeled with the thread identifier, which will be used in the sequel.

We revisit the example program of Figure 1a, with a possible speculative computation justifying $r_0 = r_1 = 0$ (omitting the occurrences):

$$(p := 1\,;\,(!q))\|(q := 1\,;\,(!p)) \xrightarrow[t_0]{\mathsf{rd}_{q,0}} (p := 1\,;\,0)\|(q := 1\,;\,(!p)) \xrightarrow[t_1]{\mathsf{rd}_{p,0}} (p := 1\,;\,0)\|(q := 1\,;\,0) \xrightarrow[t_1]{\mathsf{wr}_{q,1}} \xrightarrow[t_0]{\mathsf{wr}_{p,1}} \xrightarrow{*} 0\|0$$

Validity Condition. The speculative computations presented so far are too permissive for our purposes, since the rules do not take into account possible data dependencies present in the program. The following speculation is an example:

$$r := (!p)\,;p := tt \quad \xrightarrow{\mathsf{wr}_{p,tt}} \quad r := (!p)\,;() \quad \xrightarrow{\mathsf{rd}_{p,tt}} \quad r := tt\,;() \quad \xrightarrow{*} \quad ()$$

where we can see that the reordering of the write on p and the read on p causes the read to see a value that has been put in the memory by a write that should follow the read in normal (sequential) computations. It is clear that this speculation violates the programmers intention, and therefore, this speculation should not be permitted. Intuitively, speculations will be considered valid if they do not violate the sequential semantics of expressions, and all read own actions have a preceding write with the same value. To express that a speculation does not violate the sequential semantics of the original expression we will strongly rely on the notion of speculations that are similar up to the reordering of independent steps, a concept borrowed from the early work by Berry and Lévy in the λ-calculus [6].

To define the permutation of steps formally, which is central to our result, we need to introduce some technical machinery, which might be familiar from a "true concurrency" perspective. Let us define the residual of an occurrence o' after a step with action a at occurrence o in the expression e, which indicates where a subexpression at o' in e remains (if any) after a step a at o:

$$o'/_e(a,o) \triangleq \begin{cases} o' & \text{if } o \not\leq o',\ \text{or } o' = o \cdot (\_[]) \cdot o'' \ \&\ a = \beta_v \\ & \quad \text{or } o' = o \cdot (\lambda\_[]\_) \cdot o'' \ \&\ a = \beta_v \ \&\ e@o' \text{ is a redex} \\ o \cdot o'' & \text{if } o' = o \cdot (\lambda\_[]\_) \cdot o'' \ \&\ a = \beta \\ \mathtt{undef} & \text{otherwise} \end{cases}$$

In the following we write $o'/_e(a,o) \equiv o''$ to mean that the residual of o' after (a,o) is defined, and it is o''. Notice that if $o'/_e(a,o) \equiv o''$ with $o' \in \mathcal{O}cc^*$ then $o'' = o'$ and $o \not\leq o'$.

We now prove that if two consecutive actions are not related by redex creation (i.e. they have residuals after each other), then reordering their steps in the speculation leads to the same result. This property is key to the definition of *valid* speculations[4]:

Lemma 4 (Reordering Lemma). *If* $e_0 \xrightarrow[o_0]{a_0} e \xrightarrow[o_1]{a_1} e_1$ *with* $o_1 \equiv o_1'/_{e_0}(a_0, o_0)$ *and* $o_0' \equiv o_0/_{e_0}(a_1, o_1')$*, then there exists* e' *(unique up to* α*-conversion) such that* $e_0 \xrightarrow[o_1']{a_1} e' \xrightarrow[o_0']{a_0} e_1$.

[4] The proof of this and subsequent results are to be found in the appendices.

To take into account the dependencies in the program we need a notion of conflict. We introduce the notations $\mathcal{MR}d_p \triangleq \{\mathsf{rd}_{p,v}, \mathsf{cas}_{p,v} | v \in \mathcal{V}al\}$ for the read actions on reference p that effectively use the memory, and $\mathcal{MW}r_p \triangleq \{\mathsf{wr}_{p,v}, \mathsf{cas}_{p,tt} | v \in \mathcal{V}al\}$ for write actions that modify the memory and define then the conflict relation.

Definition 5 (Conflicting Actions). *We denote by $\#$ the following relation on actions:*

$$\# \triangleq \bigcup_{p \in \mathcal{R}ef} (\mathcal{MW}r_p \times \mathcal{MW}r_p) \cup (\mathcal{MW}r_p \times \mathcal{MR}d_p) \cup (\mathcal{MR}d_p \times \mathcal{MW}r_p)$$

Notice that we explicitly have that speculative read actions are not conflicting with write actions on the same thread. Formally: $(\mathsf{wr}_{p,v}, \mathsf{rd}^{\circ}_{p,w}) \notin \#$.

We now define a *reordering relation* establishing when two speculations correspond to each other up to the reordering of intermediate steps. This definition is parametric on a *dependency* relation $\mathcal{D}$. We only require that $\# \subseteq \mathcal{D}$.

Definition 6 (Reordering Relation). *Given a dependency relation $\mathcal{D}$ we define a reordering relation between speculations, called* $\mathcal{D}$-reordering*, to be the least preorder $\propto^{\mathcal{D}}$ such that if $e_0 \xrightarrow[o_0]{a_0} e \xrightarrow[o_1]{a_1} e_1$ with $o'_0 \equiv o_0/_e(a_1, o'_1)$ and $o'_1/_e(a_0, o_0) \equiv o_1$, and $\neg(a_0 \mathcal{D} a_1)$, then $\sigma_0 \cdot e_0 \xrightarrow[o'_1]{a_1} e' \xrightarrow[o'_0]{a_0} e_1 \cdot \sigma_1 \;\; \propto^{\mathcal{D}} \;\; \sigma_0 \cdot e_0 \xrightarrow[o_0]{a_0} e \xrightarrow[o_1]{a_1} e_1 \cdot \sigma_1$ where e' is determined by Reordering Lemma.*

A speculation will be considered valid if it is a reordering of a normal speculation. In other words, if it can be reordered to a speculation where all actions take place in program order. In addition, we will check that the actions $\mathsf{rd}^{\circ}_{p,v}$ return the last value written to p in the normal speculation, conforming the semantics of write buffering. To do so, we need to identify steps that represent the same transition in reordering related speculations. We use the notions of a *step* and *step family*, originally introduced as "redex-with-history" in [6,12].

Definition 7 (Step and Step Family). *A* step *is a pair $[\sigma, (a, o)]$ of a speculation $\sigma : e \xrightarrow{*} e'$ and an action a at occurrence o such that $e' \xrightarrow[o]{a} e''$ for some expression e''. The binary relation $\sim^{\mathcal{D}}$ on steps, meaning that two steps are in the* same family*, is the equivalence relation generated by the rule*

$$\frac{\exists \sigma''.\ \sigma' \propto^{\mathcal{D}} \sigma \cdot \sigma'' \ \text{ or } \ \sigma \cdot \sigma'' \propto^{\mathcal{D}} \sigma' \ \&\ o' \equiv o/\sigma''}{[\sigma, (a, o)] \sim^{\mathcal{D}} [\sigma', (a, o')]}$$

We now define the validity of speculations, where we see that $\mathsf{rd}^{\circ}_{p,v}$ actions take their value from the last write on p in the corresponding normal speculation.

Definition 8 (Speculation Validity). *A speculation σ is $\mathcal{D}$-valid if there is a normal speculation σ' such that $\sigma \propto^{\mathcal{D}} \sigma'$, and if $\sigma' = \sigma'_0 \cdot \xrightarrow[o]{\mathsf{rd}^{\circ}_{p,v}} \cdot \sigma'_1$ then there exists σ''_0, σ'''_0 and o' such that $\sigma'_0 = \sigma''_0 \cdot \xrightarrow[o']{\mathsf{wr}_{p,v}} \cdot \sigma'''_0$ where σ'''_0 contains no $\mathsf{wr}_{p,\_}$*

actions. We call the step $[\sigma_0'', (\mathsf{wr}_{p,v}, o')]$ *the matching write of* $[\sigma_0', (\mathsf{rd}^{\mathsf{o}}_{p,v}, o)]$*, and we denote it* $\mathsf{match}\big([\sigma_0', (\mathsf{rd}^{\mathsf{o}}_{p,v}, o)]\big)$.

We now specialize the speculative semantics to TSO and PSO, by particularizing the dependency relations that characterize them. We introduce the notations $\mathcal{R}d_p \triangleq \mathcal{MR}d_p \cup \{\mathsf{rd}^{\mathsf{o}}_{p,v} \mid v \in \mathcal{V}al\}$ and $\mathcal{W}r_p \triangleq \mathcal{MW}r_p \cup \{\mathsf{cas}_{p,ff}\}$ of read and write actions on location p (not necessarily accessing the memory as opposed to $\mathcal{MR}d_p$ and $\mathcal{MW}r_p$), and $\mathcal{R}d \triangleq \bigcup_{p\in\mathcal{R}ef} \mathcal{R}d_p$ and $\mathcal{W}r \triangleq \bigcup_{p\in\mathcal{R}ef} \mathcal{W}r_p$. As a step towards the dependencies of TSO, PSO and RMO we define the dependencies induced by barrier actions denoted by $\ltimes^{TSO}$, $\ltimes^{PSO}$ and $\ltimes^{RMO}$ where we assume the barrier actions `rr` and `rw` generated by the barriers $\langle \mathtt{rd}|\mathtt{rd}\rangle$ and $\langle \mathtt{rd}|\mathtt{wr}\rangle$ respectively which prevent the reordering of reads with subsequent reads and writes respectively, wich are allowed by RMO.

$$\begin{array}{l}
\ltimes^{TSO} \triangleq (\mathcal{W}r \times \{\mathtt{wr}\}) \cup (\{\mathtt{wr}\} \times \mathcal{R}d) \\
\ltimes^{PSO} \triangleq \ltimes^{TSO} \cup (\mathcal{W}r \times \{\mathtt{ww}\}) \cup (\{\mathtt{ww}\} \times \mathcal{W}r) \\
\ltimes^{RMO} \triangleq \ltimes^{PSO} \cup (\mathcal{R}d \times \{\mathtt{rr}, \mathtt{rw}\}) \cup (\{\mathtt{rw}\} \times \mathcal{W}r) \cup (\{\mathtt{rr}\} \times \mathcal{R}d)
\end{array}$$

Definition 9 (TSO, PSO and RMO). *The TSO, PSO and RMO memory models are characterized by the following dependency relations:*

$$\begin{array}{l}
\mathcal{D}^{RMO} \triangleq \# \cup \ltimes^{RMO} \\
\mathcal{D}^{PSO} \triangleq \# \cup \ltimes^{PSO} \cup (\mathcal{R}d \times \mathcal{R}d) \cup (\mathcal{R}d \times \mathcal{W}r) \\
\mathcal{D}^{TSO} \triangleq \# \cup \ltimes^{TSO} \cup (\mathcal{R}d \times \mathcal{R}d) \cup (\mathcal{R}d \times \mathcal{W}r) \cup (\mathcal{W}r \times \mathcal{W}r)
\end{array}$$

The semantic definition of RMO is here only given for completeness, we shall not refer to it in the rest of the paper.

Finally, we need a notion of when a write should, or should not, be considered committed (cf. write buffers). To do that we need to know when two steps in a speculation are inherently ordered; that is, they are ordered similarly for all possible valid reorderings of the speculation.

Definition 10 (Step Ordering). *Given a speculation* $\sigma = \sigma_0 \cdot \xrightarrow[o_0]{a_0} \cdot\sigma_1 \cdot \xrightarrow[o_1]{a_1} \cdot\sigma_2$*, we have* $[\sigma_0, (a_0, o_0)] \prec_\sigma [\sigma_0 \cdot \xrightarrow[o_0]{a_0} \cdot\sigma_1, (a_1, o_1)]$*, iff for all* σ' *with* $\sigma' \propto^{\mathcal{D}} \sigma$ *then* $\sigma' = \sigma_0' \cdot \xrightarrow[o_0']{a_0} \cdot\sigma_1' \cdot \xrightarrow[o_1']{a_1} \cdot\sigma_2'$ *with* $[\sigma_0, (a_0, o_0)] \sim [\sigma_0', (a_0, o_0')]$ *and* $[\sigma_0 \cdot \xrightarrow[o_0]{a_0} \cdot\sigma_1, (a_1, o_1)] \sim [\sigma_0' \cdot \xrightarrow[o_0']{a_0} \cdot\sigma_1', (a_1, o_1')]$.

Now we can define when, in a speculative computation, a write has to be considered committed. We denote by $\gamma|_t$ the projection of thread t over γ.

Definition 11. *Given* $\gamma = \gamma_0 \cdot \xrightarrow[t,o]{\mathsf{wr}_{p,v}} \cdot\gamma_1$*, the step* $[\gamma_0|_t, (\mathsf{wr}_{p,v}, o)]$ *is* committed *in* γ *if there are* γ_1'*,* γ_1''*,* t'*,* o'*,*w *such that* $\gamma_1 = \gamma_1' \cdot \xrightarrow[t',o']{\mathsf{rd}_{p,w}} \cdot\gamma_1''$ *, or* $\gamma_1 = \gamma_1' \cdot \xrightarrow[t,o']{\mathsf{wr}_{q,w}} \cdot\gamma_1''$ *with* $[\gamma_0|_t, (\mathsf{wr}_{p,v}, o)] \prec_{\gamma|_t} [\gamma_0 \cdot \xrightarrow[t,o]{\mathsf{wr}_{p,v}} \cdot\gamma_1'|_t, (\mathsf{wr}_{q,w}, o')]$ *and* $[\gamma_0 \cdot \xrightarrow[t,o]{\mathsf{wr}_{p,v}} \cdot\gamma_1', (\mathsf{wr}_{p,q}, o')]$ *is committed in* γ.

To see why we require this condition for validity, consider the following thread system in PSO where we depict only threads:

$$\begin{bmatrix} p := tt\,; \\ \langle \mathtt{wr}|\mathtt{wr}\rangle\,; \\ q := tt\,; \\ (!p) \end{bmatrix} \parallel \left[(!q)\right] \xrightarrow[t_0]{\mathsf{wr}_{p,tt}} \begin{bmatrix} \langle \mathtt{wr}|\mathtt{wr}\rangle\,; \\ q := tt\,; \\ (!p) \end{bmatrix} \parallel \left[(!q)\right] \xrightarrow[t_0]{\mathtt{ww}} \begin{bmatrix} q := tt\,; \\ (!p) \end{bmatrix} \parallel \left[(!q)\right]$$
$$\xrightarrow[t_0]{\mathsf{wr}_{q,tt}} \left[(!p)\right] \parallel \left[(!q)\right] \xrightarrow[t_1]{\mathsf{rd}_{q,tt}} \left[(!p)\right] \parallel \left[\ tt\ \right]$$

It is clear that the final read of p by t_0 cannot be a $\mathsf{rd}^{\mathsf{o}}_{p,v}$ (that is a read of an uncommitted write), since the write of q has already been made globally visible, and there is a $\langle \mathtt{wr}|\mathtt{wr}\rangle$ between the write of p and the one of q. This is obvious in the semantic with write-buffers but has to be required for speculations.

We can now give the definition of validity for TSO and PSO speculative computations.

Definition 12 (Valid Speculative Computation). *A speculative computation γ is* $\mathcal{D}$-valid *iff for every thread t we have that $\gamma|_t$ is a $\mathcal{D}$-valid speculation, and additionally, if $\gamma = \gamma' \cdot \xrightarrow[t,o']{\mathsf{rd}^{\mathsf{o}}_{p,v}} \cdot\, \gamma_2$ where $\gamma' = \gamma_0 \cdot \xrightarrow[t,o]{\mathsf{wr}_{p,v}} \cdot\, \gamma_1$, and $\mathsf{match}([\gamma'|_t, (\mathsf{rd}^{\mathsf{o}}_{p,v}, o')]) \sim [\gamma_0|_t, (\mathsf{wr}_{p,v}, o)]$ then $[\gamma_0|_t, (\mathsf{wr}_{p,v}, o)]$ is not committed in γ'.*

Hence, $\mathcal{D}^{TSO}$-valid speculative computations describe TSO, and similarly $\mathcal{D}^{PSO}$-valid speculative computations describe PSO. Thus, the examples of Figure 1 are valid.

3 A Formal Comparison

We prove that both instances of PSO are equivalent by showing how a computation with write buffers can be transformed into an equivalent one with speculations, and vice versa. A similar result for TSO is obtained as a corollary observing that the semantic rules for PSO are a superset of the rules of TSO.

Since the mechanisms used in these formalizations are very different, we introduce a *third calculus* incorporating both, write buffers and speculations. We consider this calculus merely as a tool for the proof. We then show that computations of PSO with write buffers can be embedded in this third calculus, and so can computations of PSO with speculations. Our proof of coincidence amounts to proving that: starting from a computation of the third calculus embedding a computation with write buffers (or speculations) one can reorder actions, with an appropriate instance of the reordering equivalence relation, to get a computation that is an embedding of a speculative (respectively write buffers) PSO computation. To simplify the results we disregard $\mathsf{cas}_{p,v}$ observing that its treatment can be deduced from similar conditions on read and write actions.

Let us formalize this third calculus, which we call *merge*. The semantic rules for single expressions are exactly the same as for the semantics of speculations;

that is, the rules of Figure 2 with speculation contexts $\mathbf{\Sigma}$ instead of $\mathbf{E}$. For example, the rule for read in Figure 2 is translated in the merge calculus to:

$$\mathbf{\Sigma}[(\mathbf{!}\, p)] \xrightarrow{\mathsf{rd}_{p,v}} \mathbf{\Sigma}[v]$$

To cope with speculation we further add the redex $(\lambda x e_0 e_1)$ and its associated reduction rule presented in Equation (1).

Configurations, and the rules for thread systems are the same as presented for the semantics of write buffers in Figure 4 with the exception of the rule $\mathsf{rd}^{\circ}_{p,v}$ which in merge has no constraints. As we did in the semantics of speculations the transitions will be labeled with the occurrence and the thread performing the action. Let us refresh the semantics of Figure 4, emphasizing the only change we make:

$$\frac{e \xrightarrow[o]{a} e'}{(S,(B,e_t)\|T) \xrightarrow[t,o]{a} (S',(B',e'_t)\|T)}\ (*)\ \begin{cases} a = \mathsf{rd}^{\circ}_{p,v} & \Rightarrow\ S' = S\ \&\ B' = B \\ \dots \end{cases}$$

where the conditions not mentioned are similar to the ones of $(*)$ in Figure 4.

As we have done before, we now define an equivalence by reordering of independent steps for merge, which allows us to compare its executions. Importantly the addition of buffers to the calculus of the previous section makes the definition of conflict, and hence dependency, change. This is because in merge the actual memory update is done by the buffer update rule ($\mathsf{bu}_{p,v}$) rather than the write rule ($\mathsf{wr}_{p,v}$). Moreover, memory update only affects reads from the memory ($\mathsf{rd}_{p,v}$) unlike buffer reads ($\mathsf{rd}^{\circ}_{p,v}$) which are retrieved from the thread local buffer. The definitions of conflict and dependency for the merge calculus are then:

$$\#^{MG} = \{(\mathsf{bu}_{p,v}, \mathsf{bu}_{p,w}), (\mathsf{rd}_{p,v}, \mathsf{bu}_{p,w}), (\mathsf{bu}_{p,v}, \mathsf{rd}_{p,w}) \mid p \in \mathcal{R}ef,\ v, w \in \mathcal{V}al\}$$
$$\mathcal{D}^{MG} \triangleq \#^{MG} \cup \ltimes^{PSO} \cup (\mathcal{R}d \times \mathcal{R}d) \cup (\mathcal{R}d \times \mathcal{W}r)$$

It is not hard to see that every computation of a program in the semantics of PSO with write buffers is strictly included the semantics of that program in merge. This is obvious since the configurations are the same, and the set of semantic rules of write buffer PSO is strictly included in the set of rules of merge. Similarly any computation of PSO with speculation can be trivially embedded into merge by simply forcing a buffer update (by the $\mathsf{bu}_{p,v}$ rule) after every write ($\mathsf{wr}_{p,v}$). We shall denote the merge trace resulting from the speculation trace γ by this forcing semantics by $\lceil\gamma\rceil$. Conversely, a computation γ of the merge-calculus can be related with PSO speculations by "erasing" all the buffer update actions that immediately follow its generating write. We shall denote by $\lfloor\gamma\rfloor$ this operation in the sequel. The following remark establishes these trivial embeddings.

Remark 13. Every computation $\gamma : C \xrightarrow{*} C'$ of PSO with write buffers (as in Figure 4) is also a legal execution of the merge-calculus. For every valid speculative computation $\gamma : C \xrightarrow{*} C'$ (as in Figure 5), $\lceil\gamma\rceil : C \xrightarrow{*} C'$ is a computation of the merge-calculus.

This remark provides us with embeddings and projections from, and to, the merge-calculus for both write buffers and speculations. The rest of the proof only deals with the reordering of steps in the merge-calculus. To that end, we reproduce the result of Lemma 4, this time for the merge-calculus.

Lemma 14 (Merge Reordering). *If* $e_0 \xrightarrow[o_0]{a_0} e \xrightarrow[o_1']{a_1} e_1$*, then there exists* e' *such that* $e_0 \xrightarrow[o_1]{a_1} e' \xrightarrow[o_0']{a_0} e_1$ *such that* $o_1' \equiv o_1/_{e_0}(a_0, o_0)$ *and* $o_0' \equiv o_0/_{e_0}(a_1, o_1)$.

We can then instantiate the reordering relation (Definition 6) of the previous section using the merge reordering lemma, and we denote by $\propto^{MG}$ the merge reordering relation: $\propto^{\mathcal{D}^{MG}}$.

Much can be said about the merge-calculus equivalence by reordering relation. However the merge-calculus is just a tool in our proof, with little practical interest for the memory models we consider in this paper. We point the interested reader to Appendix B.3, and more generally the full Appendix B, for a detailed account of the merge-calculus intermediate results. Suffice it to say in this section that the equivalence by reordering of merge allows us to transform, by reorderings independent steps, the embedding in merge of a trace of PSO with write-buffers (or PSO with speculations) into a an embedding in merge of a trace of PSO with speculations (or PSO with write-buffers respectively). These are the main results that we consider next.

From Buffers to Speculations. In the following theorem we show how to transform the a PSO write-buffers computation (embedded in the merge-calculus) into an equivalent merge-computation where the buffers have no effect. By this we mean that each write in the resulting computation is immediately followed by its update into memory. The resulting computation corresponds to the embedding of a PSO speculation in the merge-calculus, and therefore erasing the buffer updates we obtain a PSO speculative computation.

The intuition behind this proof is that, while writes in the write-buffer calculus are executed in program order – that is, respecting the program text –, their effects are only visible at the time when the buffer is updated into memory. Based on this observation we conclude that in the semantics with write buffers, a write does not affect the behavior of other threads until its buffer update. Therefore, we can push the write to happen at a later time, by introducing speculations that execute instructions that follow the write in program order. In fact, doing so we prove that we can postpone executing the write up to the point where its buffer update is executed. For a simplified example consider a trace γ of PSO with write buffers where we stand out an occurrence of a write and its subsequent corresponding buffer update:

$$\gamma = \gamma_0 \cdot \xrightarrow[t]{\mathsf{wr}_{p,v}} \cdot \gamma_1 \cdot \xrightarrow[t]{\mathsf{bu}_{p,v}} \cdot \gamma_2$$

The following theorem is based on an intermediate result proving that the merge segment γ_1 can be permuted by the merge reordering relation to render an

equivalent trace:

$$\gamma \propto^{MG} \gamma_0 \cdot \gamma_1' \cdot \xrightarrow[t]{\mathsf{wr}_{p,v}} \cdot \xrightarrow[t]{\mathsf{bu}_{p,v}} \cdot \gamma_1'' \cdot \gamma_2$$

where $\gamma_1' \cdot \xrightarrow[t]{\mathsf{wr}_{p,v}} \cdot \xrightarrow[t]{\mathsf{bu}_{p,v}} \cdot \gamma_1''$ has speculative behavior (as permitted by the merge calculus). Notice that all actions in γ_1 by threads other than t are independent of the write, and for actions on t we show that there is at least one such write that can be speculated upon (see the formalization in Corollary 41 in Appendix B).

The inductive application of the intuition stated above renders a computation where all writes and barrier actions are immediately followed by their corresponding commit. The proof is by induction on the number of write and barrier actions that are not immediately committed.

Theorem 15 (Write Buffers ⇒ Speculations). *For any computation $\gamma : C \xrightarrow{*} C'$ of the formalization of PSO with write-buffers there exists a merge computation $\gamma' : C \xrightarrow{*} C'$ such that for every thread t, $\gamma'|_t \propto^{MG} \gamma|_t$. Moreover, $\lfloor \gamma' \rfloor$ is a $\mathcal{D}^{PSO}$-valid speculative computation.*

From Speculations to Buffers. As the reader might expect, the converse argument follows the same idea in the opposite direction. As a simplified example, suppose that we start with a PSO speculative computation of the form

$$\gamma = \gamma_0 \cdot \xrightarrow[t]{\mathsf{wr}_{p,v}} \cdot \gamma_1$$

Our embedding of γ into the merge would render:

$$\lceil \gamma \rceil = \lceil \gamma_0 \rceil \cdot \xrightarrow[t]{\mathsf{wr}_{p,v}} \cdot \xrightarrow[t]{\mathsf{bu}_{p,v}} \cdot \lceil \gamma_1 \rceil$$

We show in Appendix B (Lemma 47) that $\lceil \gamma_0 \rceil$ can be decomposed to obtain:

$$\lceil \gamma \rceil \propto^{MG} \lceil \gamma_0' \rceil \cdot \xrightarrow[t]{\mathsf{wr}_{p,v}} \cdot \lceil \gamma_0'' \rceil \cdot \xrightarrow[t]{\mathsf{bu}_{p,v}} \cdot \lceil \gamma_1 \rceil$$

such that the occurrence of the write $\mathsf{wr}_{p,v}$ is no longer speculative – that is, respects the program order. We leave the details of this construction to Appendix B.

Using this argument inductively we conclude that given a merge calculus computation where all thread projections are valid, there is a merge computation with the same initial and final configurations such that the steps are reordered by pushing write and barrier actions to their normal occurrence – that is, respecting the program order –, and is therefore a PSO write buffer computation.

Theorem 16 (Speculations ⇒ Write Buffers). *Given $\gamma : C \xrightarrow{*} C'$ a $\mathcal{D}^{PSO}$-valid speculative computation of the formalization of PSO with speculations, there exists a merge computation $\gamma' : C \xrightarrow{*} C$ such that for all $t \in \mathcal{T}id$, $\lceil \gamma \rceil|_t \propto^{MG} \gamma'|_t$. Moreover, γ' is a computation of the calculus with buffers.*

Notice that although the proofs we provided are stated for PSO, nothing in the proof themselves is PSO specific. On the contrary, they are stated using generic notions of conflict/dependency. Therefore the same result holds for TSO.

Corollary 17. *The semantics of TSO with write buffers and speculations are equivalent.*

As a final corollary of the proof of equivalence of the semantics we get the proof of the fundamental property for the speculative semantics of TSO and PSO, which we did not prove in Section 2 nor in [8]. This illustrates the power of these different presentations of the semantics. While the proof of the fundamental property for write-buffering semantics was studied in [7], its proof for the semantics with speculations was missing in [8]. Hence, our equivalence result establishes the fundamental property for speculations without needing a new proof strategy.

Corollary 18 (Fundamental Property). *The speculative semantics of TSO and PSO satisfy the fundamental property of relaxed memory models.*

The proof is an immediate consequence of Theorems 3, 16 and 15.

4 Related Work

Our semantics of TSO and PSO with write buffers instantiates our framework [7]. In [14] a TSO-like semantics with write buffers is given for x86 architectures. This semantics is very similar to the semantics of TSO we present here, which results as a natural consequence of instantiating [7]. The speculative semantics of TSO, PSO and RMO are based on our framework of [8]. However, although the principle of speculation is the same, value-speculation was not considered in [8] which greatly simplifies, and subsumes the technical treatment of that paper. In that sense, the speculation calculus of this paper supersedes the framework of [8]. The equivalence between the formalizations of TSO and PSO in the different frameworks are new to this paper and were developed as part of the thesis [15] but are otherwise unpublished. Importantly [7,8] focus on high-level programming languages whereas in this work we focus on architectures through the use of those frameworks.

There are many formalizations of TSO, PSO and RMO in the literature, each with a specific goal in mind. Of the axiomatic definitions of these architectures we distinguish [3,18]. There are as well other operational formalizations, eg. [4,5]. Most of these leave the programming language abstract. In particular [4,5] focus on decidability rather than on the programming language semantics, and therefore the language is immaterial. Our work is unique in its focus is on a programming languages semantics and programming languages techniques for relaxed memory models.

5 Conclusion

We provided two different formalizations of the TSO and PSO memory models using different operational frameworks. We prove that these instantiations are equivalent using standard programming languages techniques based on the permutation equivalence of [6]. The more sophisticated model of speculations proves to be more general than the one of write buffers. Our proofs show the potential of operational formalizations of relaxed memory models to support technical developments. We speculate that operational models should also be well suited to support verification techniques.

References

1. S. V. Adve and K. Gharachorloo. Shared Memory Consistency Models: A Tutorial. *Computer*, 29:66–76, 1996.
2. S. V. Adve and M. D. Hill. Weak Ordering — a New Definition. In *ISCA*, pages 2–14, New York, NY, USA, 1990. ACM.
3. J. Alglave. *A Shared Memory Poetics*. PhD thesis, Université Paris 7, 2010.
4. M. F. Atig, A. Bouajjani, S. Burckhardt, and M. Musuvathi. On the verification problem for weak memory models. In *POPL*, POPL '10, pages 7–18, 2010.
5. M. F. Atig, A. Bouajjani, S. Burckhardt, and M. Musuvathi. What's decidable about weak memory models? In *ESOP*, pages 26–46, 2012.
6. G. Berry and J.-J. Lévy. Minimal and Optimal Computations of Recursive Programs. *J. ACM*, 26(1):148–175, 1979.
7. G. Boudol and G. Petri. Relaxed Memory Models: an Operational Approach. In *POPL*, pages 392–403, New York, NY, USA, 2009. ACM.
8. G. Boudol and G. Petri. A Theory of Speculative Computation. In *ESOP*, pages 165–184. LNCS, 2010.
9. C. Flanagan, A. Sabry, B. F. Duba, and M. Felleisen. The Essence of Compiling with Continuations. In *PLDI*, pages 237–247, New York, NY, USA, 1993. ACM.
10. R. Jagadeesan, C. Pitcher, and J. Riely. Generative Operational Semantics for Relaxed Memory Models. In *ESOP*, pages 307–326, 2010.
11. L. Lamport. How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programm. *IEEE Trans. Comput.*, 28(9):690–691, 1979.
12. J.-J. Lévy. Optimal Reductions in The Lambda Calculus. *in To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism, Academic Press*, pages 159–191, 1980.
13. J. Manson, W. Pugh, and S. V. Adve. The Java memory model. In *POPL '05*, pages 378–391, New York, NY, USA, 2005. ACM.
14. S. Owens, S. Sarkar, and P. Sewell. A better x86 memory model: x86-tso. In *TPHOLs*, pages 391–407, 2009.
15. G. Petri. *Operational Semantics of Relaxed Memory Models*. PhD thesis, Nice, 2010. (`http://www.cs.purdue.edu/homes/gpetri/publis/thesisPetri.pdf`).
16. G. Petri. Studying Operational Models of Relaxed Concurrency (Extended Version), 2013. (`http://www.cs.purdue.edu/homes/gpetri/publis/opsem-long.pdf`).
17. V. A. Saraswat, R. Jagadeesan, M. M. Michael, and C. von Praun. A theory of memory models. In *PPOPP*, pages 161–172, 2007.

18. S. Sarkar, P. Sewell, F. Z. Nardelli, S. Owens, T. Ridge, T. Braibant, M. O. Myreen, and J. Alglave. The Semantics of x86-CC Multiprocessor Machine Code. In *POPL*, pages 379–391, New York, NY, USA, 2009. ACM.
19. P. Sewell, S. Sarkar, S. Owens, F. Z. Nardelli, and M. O. Myreen. x86-TSO: a Rigorous and Usable Programmer's Model for x86 Multiprocessors. *CACM*, 53(7):89–97, 2010.
20. Inc. CORPORATE. SPARC. *The SPARC Architecture Manual (version 9)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1994.

Appendix A Speculation

For the following proof we will use the following remark whose proof is trivial by induction.

Remark 19. If $e \xrightarrow[o]{a} e'$ then $\{x/v\}e \xrightarrow[o]{a} \{x/v\}e'$ for any v.

We can prove the reordering lemma.

Lemma 4 (Reordering Lemma)**.** *If* $e_0 \xrightarrow[o_0]{a_0} e \xrightarrow[o_1]{a_1} e_1$ *with* $o_1 \equiv o_1'/_{e_0}(a_0, o_0)$ *and* $o_0' \equiv o_0/_{e_0}(a_1, o_1')$*, then there exists* e' *(unique up to* α*-conversion) such that*

$$e_0 \xrightarrow[o_1']{a_1} e' \xrightarrow[o_0']{a_0} e_1$$

Proof. By cases on the respective positions of o_0 and o_1'. Notice first that if $o_0 \not\leq o_1'$ and $o_1' \not\leq o_0$ (that is the occurrences are disjoint), then $o_1 = o_1' \equiv o_1/_{e_0}(a_0, o_0)$ and $o_0' = o_0 \equiv o_0/_{e_0}(a_1, o_1')$, and it is easy to see that the two speculations can be done in any order.

Let us assume that $o_0 < o_1'$. Since o_1 is well defined, according to the definition of $o_1'/_e(a_0, o_0)$ we have three possibilities:

- $o_1' = o_0 \cdot (\lambda\_[]\_) \cdot o_1''$ and $a_0 = \beta_v$ and $e@o_1'$ is a redex. In this case we have $e_0 = \mathbf{\Sigma}_0[(\lambda x e_0' \bar{e})]$ and $e = \mathbf{\Sigma}_0[(\lambda v^?(\{x \mapsto v\}e_0')\bar{e})]$. From the hypotheses we have $e_0' \xrightarrow[o_1'']{a_1} e_0''$, and using Remark 19 we have that $\{x \mapsto v\}e_0' \xrightarrow[o_1'']{a_1} \{x \mapsto v\}e_0''$. We conclude then with $e' = \mathbf{\Sigma}_0[(\lambda x(\{x \mapsto v\}e_0'')\bar{e})]$ where verifying that the steps can be commuted is trivial.
- $o_1' = o_0 \cdot (\lambda\_[]\_) \cdot o_1''$ and $a_0 = \beta$. Then we have that $e_0 = \mathbf{\Sigma}_0[(\lambda v^? e_0' v)]$ and $e = \mathbf{\Sigma}_0[e_0']$. But from the hypotheses we also have $e_0' \xrightarrow[o_1'']{a_1} e_0''$. Then we have $e' = \mathbf{\Sigma}_0[(\lambda v^? e_0'' v)]$ and the conclusion is obvious.
- $o_1' = o_0 \cdot (\_[]) \cdot o_1''$ and $a_0 = \beta_v$. Then we know that $e_0 = \mathbf{\Sigma}_0[(\lambda x e_0' \bar{e})]$ and from the hypotheses $\bar{e} \xrightarrow[o_1'']{a_1} \bar{e}'$. The candidate for e' is then $\mathbf{\Sigma}_0[(\lambda x e_0' \bar{e}')]$ and the verification that the transitions commute is immediate.

The case of $o_1' < o_0$ is symmetric, and $o_0 = o_1'$ is impossible since o_1 is well defined. □

Appendix B Correspondence Proof

We will prove that PSO executions in the semantics of write buffers correspond to executions in the semantics with speculations and vice versa. In particular, the proof also applies in a trivial way to the formalizations of TSO, and therefore we will mainly focus on PSO, which is more general.

We base our results on the reordering relation, which is a refinement of the permutations equivalence [6]. Then, we need to state the *asynchrony* lemma for the merge-calculus as we did in 4.

Lemma 14. *(Merge Reordering) If* $e_0 \xrightarrow[o_0]{a_0} e \xrightarrow[o_1']{a_1} e_1$ *with* $o_1' \equiv o_1/_{e_0}(a_0, o_0)$ *and* $o_0' \equiv o_0/_{e_0}(a_1, o_1)$ *then there exists* e' *such that* $e_0 \xrightarrow[o_1]{a_1} e' \xrightarrow[o_0']{a_0} e_1$.

Proof. The proof is almost identical to that of Lemma 4. □

The reordering relation we consider here is the instantiation of the definition 6 with the dependency relation of the merge-calculus ($\propto^{\ltimes^{MG}}$) which we shall denote by $\propto^{MG}$.

Appendix B.1 Relevant Moves

To simplify the following developments we will only consider executions in a certain normal form (Relevant Moves Normal Form) which separates the redex-creating actions at the beginning of the trace, from the memory affecting actions, which we shall call relevant moves. This way, we will not need to deal with redex creation when talking about the memory model. We can do that because we are considering an ANF calculus, where redex creation can happen at any time, by means of the β_v reduction. One can observe then, that for any computation of the merge calculus, redex creations by means of β_v reductions and reference creations ($\nu_{p,v}$) can be pushed at the beginning. All redexes and all β (i.e. speculation matching reductions) are pushed to the end of the computation. In between we find the relevant events.

The reader is invited to skip this section if not interested in the rather technical but intuitive arguments around this transformation.

We start by defining the condition that states that a speculation has its memory model related actions isolated. To that end we define the set of actions that happen in a speculation as:

$$\mathsf{act}(\sigma) = \begin{cases} \emptyset & \text{if } \sigma = \varepsilon \\ \mathsf{act}(\sigma') \cup \{a\} & \text{if } \sigma = \xrightarrow[o]{a} \cdot\, \sigma' \end{cases}$$

We can now define the *Relevant Moves Normal Form* (RMNF) for merge-calculus speculations.

Definition 20 (Relevant Moves Normal Form). *We say a speculation σ is in Relevant Moves Normal Form (RMNF) if there are σ_0, σ_1 and σ_2 such that $\sigma = \sigma_0 \cdot \sigma_1 \cdot \sigma_2$ with $\mathsf{act}(\sigma_0) \subseteq \{\beta_v, \nu_{p,v} \mid v \in \mathcal{V}al, p \in \mathcal{R}ef\}$, also $\mathsf{act}(\sigma_1) \subseteq \mathcal{R}d \cup \mathcal{W}r \cup \mathcal{B}ar$ and finally $\sigma_2 \in \beta^*$.*

Notice in the above definition that the subspeculation σ_2 contains all, and only, the memory model related actions. This is exactly the purpose of the RMNF.

Now we show that any speculation can be transformed into RMNF.

Lemma 21. *Given $e_0 \xrightarrow[o_0]{a_0} e \xrightarrow[o_1']{a_1} e_1$ such that $a_0 \notin \{\beta_v, \nu_{p,v}\}$ and $a_1 \neq \beta$ there exist o_1, o_0' and e' such that $e_0 \xrightarrow[o_1]{a_1} e' \xrightarrow[o_0']{a_0} e_1$ with $o_0' \equiv o_0/_{e_0}(a_1, o_1)$ and $o_1' \equiv o_1/_{e_0}(a_0, o_0)$.*

Proof. Notice that by the Asynchrony Lemma 14 we only need to prove that there exist o_0' and o_1 satisfying the required conditions. Let us proceed by case analysis on the relation between o_0 and o_1':

- Suppose first that $(o_0 \leq o_1')$. Then $o_1' = o_0 \cdot o''$ for some o'' and that exists $\mathbf{\Sigma}_0$ such that $e_0 = \mathbf{\Sigma}_0[r_0] \xrightarrow[o_0]{a_0} \mathbf{\Sigma}_0[\bar{e}]$ with $o_0 = @\mathbf{\Sigma}_0$. Moreover $\bar{e}$ contains a redex at o''. The only cases for a_0 such that e_0 produces a subexpression capable of containing a redex to be reduced in the next step are $a_0 \in \{\beta_v, \beta\}$ by a simple analysis on the semantic rules. Notice that by the hypothesis the only case that remains to be analyzed is $a_0 = \beta$ in which case we have $r_0 = (\lambda v^? \bar{e} v)$ for some $v \in \mathcal{V}al$ and thus $o_1 = o_0 \cdot (\lambda\_[]\_) \cdot o''$ and $o_0' = o_0$ satisfy the conditions required by the lemma.
- if $(o_0 > o_1')$ then $o_0 = o_1' \cdot o''$. If $e_0@o_1' = r_1$ (recall that this notation from section 2) is a redex then $r_1 = (\lambda v^? \bar{e} v)$ for some $\bar{e}$ and v (there is no other expression containing a redex that can be reduced in the previous step), in which case $a_1 = \beta$ contradicting the hypothesis. If $e_0@o_1'$ is not a redex then then $o_0 = o_1' \cdot (\_[])$ with again $a_1 = \beta$, a contradiction.
- if $(o_0 \not\leq o_1')$ and $(o_0 \not> o_1')$ we conclude simply with $o_0' = o_0$ and $o_1 = o_1'$.

□

Then it is not hard to see that any speculation can be reordered to obtain a RMNF equivalent speculation. In particular none of the actions in $\{\beta, \beta_v\}$ is involved in the dependency relation, which justifies the following corollary.

Corollary 22 (Relevant Moves Normal Form). *For every speculation γ there is a speculation γ' such that $\gamma' \propto^{MG} \gamma$ and γ' is in RMNF.*

Proof. The proof is trivial by reordering the actions from the left by means of Lemma 21. □

Appendix B.2 Global Relevant Moves Normal Form

From here on we do not consider the cases of "irrelevant" moves in our proofs since they are trivial. Let us now consider the obvious extension of the dependency relation to speculations, denoted by $\sigma\ \mathcal{D}^{MG}\ a$, and meaning that there exists $a' \in \mathsf{act}(\sigma)$ such that $a'\ \mathcal{D}^{MG}\ a$. Then we can prove that steps can be reordered w.r.t. *independent* subspeculations.

Lemma 23 (Independent moves). *Let $\gamma = \sigma \cdot \sigma' \cdot \sigma''$ be a RMNF computation such that σ' contains no $\{\beta_v, \beta\}$ action and let $\sigma' = \sigma_0 \cdot \xrightarrow[o]{a}$ with $\neg(\sigma_0 \mathcal{D}^{MG} a)$. Then there exist σ_0' and o' such that $\xrightarrow[o']{a} \cdot\ \sigma_0' \propto^{MG} \sigma'$ and thus $\sigma \cdot \xrightarrow[o']{a} \cdot\ \sigma_0' \cdot \sigma'' \propto^{MG} \gamma$.*

Proof. Induction in the length of σ_0. We use Lemma 21 for the inductive case and conclude by the induction hypothesis. □

We will consider buffers up to reordering of updates of different references. The equivalence of buffers is given in the following definition:

Definition 24 (Buffer equivalence). *The buffers equivalence relation is the least equivalence $\equiv$ between buffers satisfying:*

$$\frac{p \neq q}{B_0 \triangleleft [p \leftarrow v] \triangleleft [q \leftarrow w] \triangleleft B_1 \equiv B_0 \triangleleft [q \leftarrow w] \triangleleft [p \leftarrow v] \triangleleft B_1}$$

An important consequence of adding buffers to the speculative semantics is that now there are creations of transitions induced by the write-buffers; for instance, a buffer update ($\mathsf{bu}_{p,v}$) action cannot happen if the buffer is empty, and a normal read $\mathsf{rd}_{p,v}$ cannot happen if there is a pending update on reference p in the buffer, in this last case it is only after the pending writes are committed into the memory that a normal read can proceed. Simply said, there are actions that are only enabled for buffers of a certain shape. The buffer-dependency relation is a relation of two consecutive semantic steps, that clearly depends on the originating configuration. The following definition captures that intuition:

Definition 25 (Buffer Dependency). *Whenever we have $C \xrightarrow[t,o]{a} C' \xrightarrow[t,o']{a'} C''$ we say that a creates a' from C, which we denote by $(a,t) \triangleright_C (a',t)$, if the following conditions hold: $C = (S, (B,t,e) \parallel T)$ and*

$$\begin{cases} B \neq [b] \triangleright B' & \text{and } a' = \overline{b} \quad \text{or} \\ B(p) \notin (\mathtt{ww})^* & \text{and } a' = \mathsf{rd}_{p,v} \quad \text{or} \\ B(p) \neq (\mathtt{wr})^* \cdot v \cdot s' & \text{and } a' = \mathsf{bu}_{p,v} \end{cases}$$

We can now prove that whenever we have two consecutive events of the same thread in a speculative computation, such that the actions they produce are not dependent, and the events are not dependent through the buffers, then these events can happen in the reverse order in the computation, resulting in the same final configuration.

Lemma 26 (Global Reordering: Intrathread). *Given a configuration C_0 with $C_0 = (S_0, (B_0, t, e_0) \| T_0)$ such that:*

i) $C_0 \xrightarrow[t,o_0]{a_0} C \xrightarrow[t,o_1]{a_1} C_1$, *and*
ii) $a_1 \neq \beta$ *or* $a_0 \notin \{\beta_v \mid v \in \mathcal{V}al\}$, *and*
iii) $\neg(a_0 \triangleright_{C_0} a_1)$, *and*
iv) if both $a_0, a_1 \notin \{\mathsf{bu}_{p,v}, \overline{\mathtt{wr}}, \overline{\mathtt{ww}} \mid p \in \mathcal{R}ef, v \in \mathcal{V}al\}$ *then* $\neg a_0 \mathcal{D}^{MG} a_1$,

then there is C' such that:

$$C_0 \xrightarrow[t,o'_1]{a_1} C' \xrightarrow[t,o'_0]{a_0} C_1$$

Proof. Let us consider the possible cases for a_0 and a_1:

- If $a_0 = \beta$ we need to consider the following cases for a_1. If $a_1 \in \{\mathsf{bu}_{p,v}, \overline{\mathtt{wr}}, \overline{\mathtt{ww}} \mid p \in \mathcal{R}ef, v \in \mathcal{V}al\}$ we have the conclusion directly with $B' = B_1$ and $e' = e$. In the cases where $a_1 \in \mathcal{A}ct$ we have by hypothesis *ii)* and Lemma 21 that there is e' with $e_0 \xrightarrow[o'_1]{a_1} e' \xrightarrow[o'_0]{a_0} e_1 \propto^{MG} e_0 \xrightarrow[o_0]{a_0} e \xrightarrow[o_1]{a_1} e_1$ and clearly $S' = S_1$ and $B' = B_1$ since a_0 does not modify neither the store nor the buffers.
  Almost the same reasoning applies to all the cases for a_0 with $a_1 \in \{\beta_v \mid v \in \mathcal{V}al\}$ which we will not develop in the sequel.
- If both $a_0, a_1 \notin \{\beta, \beta_v, \mathsf{bu}_{p,v}, \overline{\mathtt{wr}}, \overline{\mathtt{ww}} \mid p \in \mathcal{R}ef, v \in \mathcal{V}al\}$ we simply apply the hypothesis *iv)* and the Local Asynchrony Lemma 14 to obtain the appropriate e'. Obviously the store does not change in any of these steps, and it is easy to verify that the resulting buffer is the same up to the buffer equivalence $\simeq$.
- If $a_0 = \mathsf{rd}_{p,v}$ we have, from the semantics, that $B(p) = (\mathtt{ww})^*$ and $S(p) = v$. Clearly in this case by hypothesis *iii)* we have $a_1 \notin \{\overline{\mathtt{wr}}, \mathsf{bu}_{p,w} \mid w \in \mathcal{V}al\}$ and thus the conclusion is very easy. Notice that the case where $a_1 = \mathsf{wr}_{p,w}$ has already been discarded in the previous case.
- If $a_0 = \mathsf{rd}^{\circ}_{p,v}$ the conclusion is immediate since this action does not depend nor modify in any way the buffers or the store.
- If $a_0 = \mathsf{wr}_{p,v}$ the conclusion is simple as well observing that if $a_1 = \mathsf{bu}_{p,w}$ then $B(p) = (\mathtt{wr})^* \cdot [p \mapsto w] \cdot s$ for some s; otherwise we would violate the hypothesis *ii)*. Also it is clear that $a_1 \neq \overline{\mathtt{ww}}$. We obtain the conclusion easily.
- If $a_0 \in \{\mathtt{ww}, \mathtt{wr}\}$ we have the that, given the condition *iii)*, the requirement for performing a_0 is such that the reordering is guaranteed. For instance if $a_0 = \mathtt{ww}$ and $a_1 = \overline{\mathtt{ww}}$ then $B = \mathtt{ww} \cdot s$ for some s. Similarly for $\mathtt{wr}$.
- If $a_0 = \mathsf{bu}_{p,v}$ then $a_1 \notin \{\overline{\mathtt{ww}}, \mathsf{bu}_{p,w}, \mathsf{rd}_{p,w} \mid w \in \mathcal{V}al\}$; and if $a_1 = \overline{\mathtt{wr}}$ by *iii)* we know that $B(p) = \mathtt{wr} \cdot s$ for some s, the conclusion is immediate in the remaining cases.
- If $a_0 = \overline{\mathtt{ww}}$ then $a_1 \notin \{\mathsf{bu}_{q,w}, \overline{\mathtt{wr}} \mid r \in \mathcal{R}ef,\ w \in \mathcal{V}al\}$. Again in this case the conclusion is direct. The same reasoning applies to $a_0 = \overline{\mathtt{wr}}$ where we know by *iii)* that $a_1 \neq \mathsf{rd}_{q,w}$ for all q and w, nor $a_1 = \overline{\mathtt{ww}}$.

□

And a similar, but simpler result can be derived for the case in which the events occur in different threads.

Lemma 27 (Global Reordering). *Let* $C_0 = (S, (B_0, t_0, e_0) \| (B_1, t_1, e_1) \| T)$ *and* $C_0 \xrightarrow[t_0,o_0]{a_0} C \xrightarrow[t_1,o_1]{a_1} C_1$ *with* $t_0 \neq t_1$ *and* $\neg(a_0 \ \#^{MG} \ a_1)$. *Then there exists* C' *such that*

$$C_0 \xrightarrow[t_1,o_1]{a_1} C' \xrightarrow[t_0,o_0]{a_0} C_1$$

Proof. We proceed by case analysis on a_0 and a_1. We consider only the cases of actions that access the memory, the other being trivial (Notice that $\mathsf{wr}_{p,v}$ actions do not directly access the memory and thus are trivial too):

- $a_0 = \mathsf{rd}_{p,v}$. If:
  - $a_1 = \mathsf{rd}_{q,w}$ the conclusion is trivial, even if $p = q$ (with $w = v$).
  - $a_1 = \mathsf{bu}_{q,w}$. If $p = q$ then $a_0 \# a_1$ contradicting the hypothesis. In case $p \neq q$ the conclusion is immediate.
  - Notice that $a_1 = \mathsf{wr}_{q,w}$ does not modify the memory and thus is trivial.
- $a_0 = \mathsf{bu}_{p,v}$. If:
  - $a_1 = \mathsf{rd}_{q,w}$. If $p = q$ we have a contradiction to the hypothesis and if $p \neq q$ the conclusion is immediate.
  - $a_1 = \mathsf{bu}_{q,w}$. Then $p = q \Rightarrow (a_0 \# a_1)$ and $p \neq q$ the conclusion is trivial.

□

By means of this lemma we can relate the RMNF of speculations with global computations.

Proposition 28 (Global RMNF).

Given a global computation γ *with* $\gamma = (C_i \xrightarrow[t_i]{a_i,o_i} C_{i+1})_{0 \leq i \leq n}$ *there exists an execution* γ' *starting from* C_0 *and ending in* C_{n+1} *such that* $\gamma'|_t \ \propto^{MG} \ \gamma|_t$ *and* $\gamma'|_t$ *is in RMNF for all* $t \in \mathcal{T}id$.

Proof. The proof is trivial by repeatedly applying Lemma 27 and Lemma 21. □

Appendix B.3 Step Ordering Analysis

For the results that follow we will need to identify events that are necessarily ordered (akin to causality) by the dependencies induced by the memory model. The reader can observe that *relevant* moves that cannot be reordered by the reordering relation are somehow related in a dependency chain. We establish the following ordering definition between steps.

Definition 29 (Step ordering).

Given a speculation σ, *such that* $\sigma = \sigma_0 \cdot \xrightarrow[o_0]{a_0} \cdot \sigma_1 \cdot \xrightarrow[o_1]{a_1} \cdot \sigma_2$, *we say that the step* $[\sigma_0, (a_0, o_0)]$ *is ordered before the event* $[\sigma_0 \cdot \xrightarrow[o_0]{a_0} \cdot \sigma_1, (a_1, o_1)]$, *which we shall denote* $[\sigma_0, (a_0, o_0)] \prec_\sigma [\sigma_0 \cdot \xrightarrow[o_0]{a_0} \cdot \sigma_1, (a_1, o_1)]$, *iff for all* σ' *with* $\sigma' \propto^{MG} \sigma$

then $\sigma' = \sigma'_0 \cdot \xrightarrow[o'_0]{a_0} \cdot \sigma'_1 \cdot \xrightarrow[o'_1]{a_1} \cdot \sigma'_2$ *with* $[\sigma_0, (a_0, o_0)] \sim [\sigma'_0, (a_0, o'_0)]$ *and* $[\sigma_0 \cdot \xrightarrow[o_0]{a_0} \cdot \sigma_1, (a_1, o_1)] \sim [\sigma'_0 \cdot \xrightarrow[o'_0]{a_0} \cdot \sigma'_1, (a_1, o'_1)]$.

Notice that here we are using the step equivalence $\sim$ that was defined in 7.

We can immediately observe that if in a computation we have two dependent actions, in every reordering of that computation the events are in the same order. In particular steps with conflicting actions are step ordering related.

Remark 30 (Dependency implies Ordering).

Given a speculation γ, such that $\gamma = (e_i \xrightarrow[o_i]{a_i} e_{i+1})_{0 \leq i \leq n}$ where $a_j \mathcal{D}^{MG} a_h$ with $1 \leq j < h \leq n$ we have $[\sigma_{j-1}, (a_j, o_j)] \prec_\gamma [\sigma_{h-1}, (a_h, o_h)]$.

Proof. The proof is trivial by induction on n and the definition of the reordering relation $\propto^{MG}$. □

Conversely, an event following a write, such that they are ordered, indicates that the second event is conflicting with the write, unless they are related by redex creation.

Remark 31 (Ordering implies Dependency). Given a speculation σ such that $\sigma = \sigma_0 \cdot \xrightarrow[o]{\mathsf{wr}_{p,v}} \cdot \xrightarrow[o_1]{a_1} \cdot \sigma_1$, if $[\sigma_0, (\mathsf{wr}_{p,v}, o)] \prec_\sigma [\sigma_0 \cdot \xrightarrow[o]{\mathsf{wr}_{p,v}}, (a_1, o_1)]$ with $a_1 \neq \beta$ then $\mathsf{wr}_{p,v} \mathcal{D}^{MG} a_1$.

Proof. The proof is immediate by contradiction. □

We can now prove that if two events in a RMNF speculation are not related, there must be an equivalent speculation where these events are adjacent and in the opposite order.

Lemma 32. *Given a merge-calculus speculation* σ *such that* $\sigma = \sigma_0 \cdot \xrightarrow[o_0]{a_0} \cdot \sigma_1 \cdot \xrightarrow[o_1]{a_1} \cdot \sigma_2$ *and such that* σ *is in RMNF and* $a_0, a_1 \notin \{\beta, \beta_v, \nu_{p,v}, \}$ *and also* $\neg([\sigma_0, (a_0, o_0)] \prec_\sigma [\sigma_0 \cdot \xrightarrow[o_0]{a_0} \cdot \sigma_1, (a_1, o_1)])$, *then there exist* σ'_1 *and* σ''_1 *such that:*

$$\sigma_0 \cdot \sigma'_1 \cdot \xrightarrow[o'_1]{a_1} \cdot \xrightarrow[o'_0]{a_0} \cdot \sigma''_1 \cdot \sigma_2 \propto^{MG} \sigma$$

Proof. The proof proceeds by induction on the length of σ_1:

- In the base case $\sigma_1 = \epsilon$ and thus $\sigma = \sigma_0 \cdot \xrightarrow[o_0]{a_0} \cdot \xrightarrow[o_1]{a_1} \cdot \sigma_2$ and by Remark 30 we know $\neg(a_0 \mathcal{D}^{MG} a_1)$, and thus we can apply Lemma 14 to conclude.
- In the case where $\sigma_1 = \xrightarrow[o_2]{a_2} \cdot \sigma'''_1$ we proceed by cases:

  • If $\neg(a_0 \mathcal{D}^{MG} a_2)$ we can simply apply the asynchrony lemma (14) to obtain $\xrightarrow[o'_2]{a_2} \cdot \xrightarrow[o'_0]{a_0} \cdot \sigma'''_1 \propto^{MG} \xrightarrow[o_0]{a_0} \cdot \sigma_1$. with σ'''_1 having a shorter length than σ_1 we conclude by the induction hypothesis.

• If $a_0 \mathcal{D}^{MG} a_2$ then $\neg([\sigma_0 \cdot \xrightarrow[o_0]{a_0}, (a_2, o_2)] \prec_\sigma [\sigma_0 \cdot \xrightarrow[o_0]{a_0} \cdot \sigma_1, (a_1, o_1)])$, otherwise we would have a contradiction with the hypothesis about the ordering of a_0 and a_1. Thus we can apply the induction hypothesis on the step $[\sigma_0 \cdot \xrightarrow[o_0]{a_0}, (a_2, o_2)]$ to obtain $\widehat{\sigma_1'}$ and $\widehat{\sigma_1''}$ with $\sigma_0 \cdot \xrightarrow[o_0]{a_0} \cdot \widehat{\sigma_1'} \cdot \xrightarrow[o_1'']{a_1} \cdot \xrightarrow[o_2']{a_2} \cdot \widehat{\sigma_1''} \cdot \sigma_2 \propto^{MG} \sigma$, where clearly $\widehat{\sigma_1'}$ has a shorter length than σ_1 which allows us to use the induction hypothesis to conclude. □

Also, if two events are ordered by the step ordering relation, either their actions are dependent or there is an intermediate event that is conflicting with the first one and ordered with the second one.

Lemma 33. *Given $\gamma = \sigma_0 \cdot \xrightarrow[o_0]{a_0} \cdot \sigma_1 \cdot \xrightarrow[o_1]{a_1} \cdot \sigma_2$ a RMNF execution with $a_0, a_1 \notin \{\beta, \beta_v\}$ and $[\sigma_0, (a_0, o_0)] \prec_\gamma [\sigma_0 \cdot \xrightarrow[o_0]{a_0} \cdot \sigma_1, (a_1, o_1)]$ we have one of the following:*

i) $a_0 \mathcal{D}^{MG} a_1$, or
ii) there are σ_1', σ_1'', a_2 and o_2 such that $\sigma_1 = \sigma_1' \cdot \xrightarrow[o_2]{a_2} \cdot \sigma_1''$ and $a_0 \mathcal{D}^{MG} a_2$ and $[\sigma_0 \cdot \xrightarrow[o_0]{a_0} \cdot \sigma_1', (a_2, o_2)] \prec_\gamma [\sigma_0 \cdot \xrightarrow[o_0]{a_0} \cdot \sigma_1, (a_1, o_1)]$

Proof. The proof is by induction on the length of σ_1. In the base case the conclusion is obvious satisfying the condition *i)*. In the induction case $\sigma_1 = \xrightarrow[o_3]{a_3} \cdot \widehat{\sigma_1}$. Clearly if $\neg(a_0 \mathcal{D}^{MG} a_3)$ we use Lemma 23 to reorder them and conclude by means of the induction hypothesis. If $(a_0 \mathcal{D}^{MG} a_3)$ and $[\sigma_0 \cdot \xrightarrow[o_0]{a_0}, (a_3, o_3)] \prec_\gamma [\sigma_0 \cdot \sigma_1, (a_1, o_1)]$ we have the conclusion directly. Let us suppose then that $\neg[\sigma_0 \cdot \xrightarrow[o_0]{a_0}, (a_3, o_3)] \prec_\gamma [\sigma_0 \cdot \sigma_1, (a_1, o_1)]$. In this case we can apply Lemma 32 to obtain $\widehat{\sigma_1'}$, $\widehat{\sigma_1''}$, o_1' and o_3' such that $\gamma \propto^{MG} \sigma_0 \cdot \xrightarrow[o_0]{a_0} \cdot \widehat{\sigma_1'} \cdot \xrightarrow[o_1']{a_1} \cdot \xrightarrow[o_3']{a_3} \cdot \widehat{\sigma_1''} \cdot \sigma_2$, with $\widehat{\sigma_1'}$ a shorter speculation than $\xrightarrow[o_3]{a_3} \cdot \sigma_1$, and hence we conclude by the induction hypothesis. □

The following lemmas state that some particular cases of events related by the step reordering relation, whose actions are not dependent as per the memory model reordering relation, have intermediate events that transitively relate them.

Lemma 34. *Given am RMNF speculation σ such that $\sigma = \sigma_0 \cdot \xrightarrow[o_0]{\mathsf{wr}_{p,v}} \cdot \sigma_1 \cdot \xrightarrow[o_1]{a_1} \cdot \sigma_2$ and $[\sigma_0, (\mathsf{wr}_{p,v}, o_0)] \prec_\sigma [\sigma_0 \cdot \xrightarrow[o_0]{\mathsf{wr}_{p,v}} \cdot \sigma_1, (a_1, o_1)]$ then either:*

i) $\mathsf{wr}_{p,v} \mathcal{D}^{MG} a_1$, or
ii) $\sigma_1 = \sigma_1' \cdot \xrightarrow[o']{b} \cdot \sigma_1'' \cdot \xrightarrow[o_2]{a_2} \cdot \sigma_1'''$ with $b \in \mathcal{B}ar$ (where we consider possible that $\xrightarrow[o_2]{a_2} \cdot \sigma_1''' = \varepsilon$, in which case a_1 stands for a_2), and $b \; \mathcal{D}^{PSO} \; a_2$.

Proof. The proof is by induction on the length of σ_1. Clearly if $\sigma_1 = \varepsilon$ then $\mathsf{wr}_{p,v}\mathcal{D}^{MG}a_1$ by Remark 31. Let us consider the induction case now. Let us assume that $\neg(\mathsf{wr}_{p,v}\mathcal{D}^{MG}a_1)$, otherwise we have the conclusion. We can then apply Lemma 33 to conclude that $\sigma_1 = \overline{\sigma_1} \cdot \xrightarrow[o_2]{a_2} \cdot \overline{\sigma_1'}$ with $\mathsf{wr}_{p,v}\mathcal{D}^{MG}a_2$ and $[\sigma_0 \cdot \xrightarrow[o_0]{\mathsf{wr}_{p,v}} \cdot \overline{\sigma_1}, (a_2, o_2)] \prec_\sigma [\sigma_0 \cdot \xrightarrow[o_0]{\mathsf{wr}_{p,v}} \cdot \sigma_1, (a_1, o_1)]$. Let us now consider the cases for a_2:

- if $a_2 = \mathsf{wr}_{q,w}$ for some $q \in \mathcal{R}ef$ and $w \in \mathcal{V}al$ we can apply the induction hypothesis considering $[\sigma_0 \cdot \xrightarrow[o_0]{\mathsf{wr}_{p,v}} \cdot \overline{\sigma_1}, (\mathsf{wr}_{q,w}, o_2)]$ in the place of $[\sigma_0, (\mathsf{wr}_{p,v}, o)]$, which renders the conclusion.
- if $a_2 = \mathtt{ww}$ we consider the following cases for a_1:
  - with $a_1 = \mathsf{wr}_{q,w}$ for some q and w we obtain the conclusion.
  - with $a_1 \in \{\mathsf{rd}_{q,w}, \mathsf{rd}^{\circ}_{q,w} \mid q \in \mathcal{R}ef, w \in \mathcal{V}al\}$ we can consider using Lemma 33 again to obtain that $\overline{\sigma_1'} = \widehat{\sigma_1} \cdot \xrightarrow[o_3]{a_3} \cdot \widehat{\sigma_1'}$ and $a_2\mathcal{D}^{MG}a_3$ which implies that $a_3 = \mathsf{wr}_{q,w}$ for some q and w. Moreover

$$[\sigma_0 \cdot \xrightarrow[o]{\mathsf{wr}_{p,v}} \cdot \overline{\sigma_1} \cdot \xrightarrow[o_2]{a_2} \cdot \widehat{\sigma_1}, (a_3, o_3)] \prec_\sigma [\sigma_0 \cdot \xrightarrow[o]{\mathsf{wr}_{p,v}} \cdot \sigma_1, (a_1, o_1)]$$

    This concludes the case.
- if $a_2 = \mathtt{wr}$, then we consider the following cases for a_1:
  - with $a_1 \in \{\mathsf{rd}_{q,w}, \mathsf{rd}^{\circ}_{q,w} \mid q \in \mathcal{R}ef, w \in \mathcal{V}al\}$ we have the conclusion of the lemma.
  - with $a_1 = \mathsf{wr}_{q,w}$ for some q and w we can apply Lemma 33 to obtain that $\overline{\sigma_1'} = \widehat{\sigma_1} \cdot \xrightarrow[o_3]{a_3} \cdot \widehat{\sigma_1'}$ and $\mathtt{wr}\mathcal{D}^{MG}a_3$ which implies that $a_3 \in \{\mathsf{rd}_{r,v'}, \mathsf{rd}^{\circ}_{r,v'} \mid r \in \mathcal{R}ef, v' \in \mathcal{V}al\}$ which renders the conclusion. Moreover $[\sigma_0 \cdot \xrightarrow[o]{\mathsf{wr}_{p,v}} \cdot \overline{\sigma_1} \cdot \xrightarrow[o_2]{a_2} \cdot \widehat{\sigma_1}, (a_3, o_3)] \prec_\sigma [\sigma_0 \cdot \xrightarrow[o]{\mathsf{wr}_{p,v}} \cdot \sigma_1, (a_1, o_1)]$.

□

Lemma 35. *Given a speculation* $\sigma = \sigma_0 \cdot \xrightarrow[o]{\mathsf{wr}_{p,v}} \cdot \sigma_1 \cdot \xrightarrow[o']{a} \cdot \sigma_2$ *with* $a \in \{\mathsf{rd}_{q,w}, \mathsf{rd}^{\circ}_{q,w} \mid p \neq q\}$, *or* $a = \mathsf{rd}^{\circ}_{p,w}$, *and where there are no* `wr` *actions in* σ_1 *and* $[\sigma_0, (\mathsf{wr}_{p,v}, o)] \prec_\sigma [\sigma_0 \cdot \xrightarrow[o]{\mathsf{wr}_{p,v}} \cdot \sigma_1, (a, o')]$. *Then there exist* σ_1', σ_1'' *and* σ_1''' *such that* $\sigma_1 = \sigma_1' \cdot \xrightarrow[o_0]{\mathsf{wr}_{r,w}} \cdot \sigma_1'' \cdot \xrightarrow[o_1]{\mathsf{rd}_{r,w}} \cdot \sigma_1'''$.

Proof. The proof is by induction on the length of σ_1, with the base case being vacuously true since we assume that $[\sigma_0, (\mathsf{wr}_{p,v}, o)] \prec_\sigma [\sigma_0 \cdot \xrightarrow[o]{\mathsf{wr}_{p,v}} \cdot \sigma_1, (a, o')]$. Let us consider the inductive case. We have $\neg(\mathsf{wr}_{p,v} \ltimes^{MG} a)$, and thus by Lemma 33 there must be the case that $\sigma_1 = \delta \cdot \xrightarrow[o_2]{a_2} \cdot \delta'$ with $\mathsf{wr}_{p,v}\mathcal{D}^{MG}a_2$ and $[\sigma_0 \cdot \xrightarrow[o]{\mathsf{wr}_{p,v}} \cdot \delta, (a_2, o_2)] \prec_\sigma [\sigma_0 \cdot \xrightarrow[o]{\mathsf{wr}_{p,v}} \cdot \sigma_1, (a, o')]$. Let us consider the cases for a_2 such that $\mathsf{wr}_{p,v}\mathcal{D}^{MG}a_2$ is satisfied:

– if $a_2 = \mathsf{rd}_{p,v'}$ we have the conclusion with $r = p$, $\sigma_1' = \varepsilon$ and taking $[\sigma_0, (\mathsf{wr}_{p,v}, o)]$ for the write event.
– if $a_2 = \mathsf{wr}_{r,v'}$ we have from Lemma 33 that $[\sigma_0 \cdot \xrightarrow[o]{\mathsf{wr}_{p,v}} \cdot \delta, (\mathsf{wr}_{r,v'}, o_2)] \prec_\sigma [\sigma_0 \cdot \xrightarrow[o]{\mathsf{wr}_{p,v}} \cdot \sigma_1, (a, o')]$, and thus we can apply the induction hypothesis to conclude.
– if $a_2 = \mathtt{ww}$ then can apply again Lemma 33 to obtain that $\delta' = \delta_0 \cdot \xrightarrow[o_3]{a_3} \cdot \delta_1$ with $\mathtt{ww}\mathcal{D}^{MG}a_3$ which implies that $a_3 = \mathsf{wr}_{r,v'}$ for some r and v'. Once more $[\sigma_0 \cdot \xrightarrow[o]{\mathsf{wr}_{p,v}} \cdot \delta \cdot \xrightarrow[o_2]{a_2} \cdot \delta_0, (a_3, o_3)] \prec_\sigma [\sigma_0 \cdot \xrightarrow[o]{\mathsf{wr}_{p,v}} \cdot \sigma_1, (a, o')]$ so we can apply the induction hypothesis to conclude.

□

Corollary 36. *Given a speculation* $\sigma = \sigma_0 \cdot \xrightarrow[o]{\mathtt{ww}} \cdot \sigma_1 \cdot \xrightarrow[o']{a} \cdot \sigma_2$ *with* $a \in \{\mathsf{rd}_{q,v}, \mathsf{rd}^{\circ}_{q,w} \mid p \neq q\}$ *or* $a = \mathsf{rd}^{\circ}_{p,w}$*, and where there are no* $\mathtt{wr}$ *actions in* σ_1 *and* $[\sigma_0, (\mathtt{ww}, o)] \prec_\sigma [\sigma_0 \cdot \xrightarrow[o]{\mathtt{ww}} \cdot \sigma_1, (a, o')]$*. Then there exist* σ_1'*,* σ_1'' *and* σ_1''' *such that* $\sigma_1 = \sigma_1' \cdot \xrightarrow[o_0]{\mathsf{wr}_{r,w}} \cdot \sigma_1'' \cdot \xrightarrow[o_1]{\mathsf{rd}_{r,v'}} \cdot \sigma_1'''$.

Proof. The proof is trivial applying Lemma 33 and Lemma 35. □

Lemma 37. *Let* $\sigma = \sigma_0 \cdot \xrightarrow[o]{\mathsf{wr}_{p,v}} \cdot \sigma_1 \cdot \xrightarrow[o']{\mathsf{wr}_{q,w}} \cdot \sigma_2$ *where* σ_1 *contains no* $\mathtt{ww}$ *action, with* $p \neq q$ *and* $[\sigma_0, (\mathsf{wr}_{p,v}, o)] \prec_\sigma [\sigma_0 \cdot \xrightarrow[o]{\mathsf{wr}_{p,v}} \cdot \sigma_1, (\mathsf{wr}_{q,w}, o')]$*. Then* $\sigma_1 = \sigma_1' \cdot \xrightarrow[o']{\mathsf{rd}_{p,v'}} \cdot \sigma_1''$.

Proof. The proof is by induction on the length of σ_1, with the base case being vacuous since $\sigma_1 = \epsilon$ contradicts $[\sigma_0, (\mathsf{wr}_{p,v}, o)] \prec_\sigma [\sigma_0 \cdot \xrightarrow[o]{\mathsf{wr}_{p,v}} \cdot \sigma_1, (\mathsf{wr}_{q,w}, o')]$. Let us consider the inductive case. We have $\neg(\mathsf{wr}_{p,v} \ltimes^{MG} \mathsf{wr}_{q,w})$ if $p \neq q$, and thus by Lemma 33 there must be the case that $\sigma_1 = \delta \cdot \xrightarrow[o_2]{a_2} \cdot \delta'$ with $(\mathsf{wr}_{p,v}\mathcal{D}^{MG}a_2)$ and $[\sigma_0 \cdot \xrightarrow[o]{\mathsf{wr}_{p,v}} \cdot \delta, (a_2, o_2)] \prec_\sigma [\sigma_0 \cdot \xrightarrow[o]{\mathsf{wr}_{p,v}} \cdot \sigma_1, (\mathsf{wr}_{q,w}, o')]$. Let us consider the cases for a_2 such that $\mathsf{wr}_{p,v}\mathcal{D}^{MG}a_2$ is satisfied: if $a_2 = \mathsf{rd}_{p,v'}$ we have the conclusion, and if $a_2 = \mathsf{wr}_{p,v'}$ we apply the induction hypothesis. We have from the hypothesis that $\mathtt{ww}$ does not occur in σ_1 so this concludes the lemma. □

Appendix B.4 From Write-Buffers to Speculations

It is fairly straightforward to see that for every computation of PSO as given by the semantics of write-buffers the exact same computation is a computation of the merge-calculus. This was previously stated as Remark 13, but we restate it here for clarity.

Remark 13. *Any computation* $\gamma : C \xrightarrow{*} C'$ *of PSO as provided by the semantics of write-buffers (of Figure 4) is a legal execution of the merge-calculus as well.*

We will call these computations of the merge-calculus *purely buffered*, since the only relaxation is provided by means of buffers and not speculations.

To prove our correspondence result, we need to construct a speculative computation that simulates the one with buffers. However, in the semantics of the merge-calculus the requirements for $\mathsf{rd}^{\circ}_{p,v}$ actions are almost vacuous, whereas in the semantics with write-buffers these actions can only happen under some conditions regarding the buffers. Indeed, the conditions required in the semantics with write-buffers are important to prove the correspondence of the semantics. Our proof proceeds by showing how actions can be reordered to reach a computation of the merge-calculus that corresponds to a speculative computation. In order to prove that we can reorder the actions we use the fact that they were generated by a valid computation of the semantics with buffers. For that purpose we define a property on computations of the merge-calculus that indicates that actions that have not yet been reordered do comply with the semantics of write-buffersc.

Definition 38 (Buffer Compliance).

We say that the computation γ complies with the semantics of buffers, denoted by $\mathsf{WB}(\gamma)$, if whenever $\gamma = \gamma_0 \cdot \xrightarrow[t,o]{\mathsf{wr}_{p,v}} \cdot \gamma_1 \cdot (C \xrightarrow[t,o']{\mathsf{rd}^{\circ}_{p,v}} C') \cdot \gamma_2$ with $C = (S, (B, t, e) \| T)$ and $\mathsf{match}[(\gamma_0 \cdot \xrightarrow[t,o]{\mathsf{wr}_{p,v}} \cdot \gamma_1)|_t, (\mathsf{rd}^{\circ}_{p,v}, o')] = [\gamma_0|_t, (\mathsf{wr}_{p,v}, o)]$ then $B(p) = s \cdot v \cdot \mathtt{ww}^n$ and $\mathtt{wr}$ does not occur in s.

In other words $\mathsf{rd}^{\circ}_{p,v}$ actions respect the semantics of write-buffers. Importantly the condition $\mathsf{WB}(\gamma)$ only requires that the value read be the last value in the buffer for own reads that follow their matching write in γ. It is easy to see that this condition is satisfied for every computation of the merge-calculus that is a computation of the semantics of PSO with write-buffers (Figure 4). Again, we restate this direction of Remark 13.

Remark 13. *Given a computation $\gamma : C \xrightarrow{*} C'$ of PSO as provided by the semantics of write-buffers (as in Figure 4) we have $\mathsf{WB}(\gamma)$.*

Evidently every purely buffered computation γ satisfies $\mathsf{WB}(\gamma)$.

The following definition states that in a prefix of the computation every buffer update, or committed barrier is immediately preceded by the write, or a barrier action that justifies it.

Definition 39 (Late-commit freedom). *We say a speculative computation γ is late-commit free, if $\gamma = \sigma_0 \cdot (C_0 \xrightarrow[t_0,o_0]{a_0} C \xrightarrow[t_1]{a_1} C_1) \cdot \sigma_1$ implies that:*

$$\begin{cases} a_1 = \mathsf{bu}_{p,v} \Rightarrow t_0 = t_1 \ \&\ a_0 = \mathsf{wr}_{p,v} \ \& \\ \qquad\qquad C_0 = (S, (B, t, e) \| T) \Rightarrow B(p) = \epsilon \\ a_1 = \overline{\mathtt{ww}} \ \Rightarrow t_0 = t_1 \ \&\ a_0 = \mathtt{ww} \ \&\ C_0 = (S, (\epsilon, t, e) \| T) \\ a_1 = \overline{\mathtt{wr}} \ \Rightarrow t_0 = t_1 \ \&\ a_0 = \mathtt{wr} \ \&\ C_0 = (S, (\epsilon, t, e) \| T) \end{cases}$$

We can prove a lemma that shows that events that depend on a write event can be permuted after the buffer update that corresponds to the write being

considered. This will later enable us to "move" the write action next to its buffer update. Once all writes immediately precede their corresponding update we have an execution that is similar to a speculative one.

Lemma 40 (Delayed Dependencies). *Let γ be a RMNF computation satisfying $\mathsf{WB}(\gamma)$. Suppose that $\gamma = \gamma_0 \cdot \xrightarrow[t,o]{\mathsf{wr}_{p,v}} \cdot \gamma_1 \cdot \xrightarrow[t,\varepsilon]{\mathsf{bu}_{p,v}} \cdot \gamma_2$, where $\gamma_0 \cdot \xrightarrow[t,o]{\mathsf{wr}_{p,v}} \cdot \gamma_1$ is the longest late commit free prefix of γ. Suppose as well that $[\gamma_0|_t, (\mathsf{wr}_{p,v}, o)]$ is the first uncommitted write to p by t in γ. Let γ_1' and γ_1'' be such that $\gamma_1 = \gamma_1' \cdot \xrightarrow[t,o_1]{a_1} \cdot \gamma_1''$ with $[\gamma_0|_t, (\mathsf{wr}_{p,v}, o)] \prec_{\gamma|_t} [\gamma_0 \cdot \xrightarrow[o]{\mathsf{wr}_{p,v}} \cdot \gamma_1', (a_1, o_1)]$ and $\neg(a_1 \# \gamma_1''|_t)$. Then there exists $\widehat{\gamma_1}$ such that*

$$\gamma' = \gamma_0 \cdot \xrightarrow[t,o]{\mathsf{wr}_{p,v}} \cdot \gamma_1' \cdot \widehat{\gamma_1} \cdot \xrightarrow[t,\varepsilon]{\mathsf{bu}_{p,v}} \cdot \xrightarrow[t,o_1']{a_1} \cdot \gamma_2$$

and for all $t' \neq t$ we have $\gamma'|_t' = \gamma|_t$ and $\gamma'|_t \propto^{MG} \gamma|_t$. Moreover we have $\mathsf{WB}(\gamma')$.

Proof. By induction on the size of γ_1''.

- In the base case we have $\gamma = \gamma_0 \cdot \xrightarrow[t,o]{\mathsf{wr}_{p,v}} \cdot \gamma_1' \cdot \xrightarrow[t,o_1]{a_1} \cdot \xrightarrow[t,\varepsilon]{\mathsf{bu}_{p,v}} \cdot \gamma_2$. Here we clearly have that $a_1 \neq \mathsf{rd}_{p,v'}$ for the semantics disallows $a_1 = \mathsf{rd}_{p,v'}$, and we notice that Lemma 26 allows to reorder a_1 and $\mathsf{bu}_{p,v}$ provided that $\neg(a_1 \triangleright_C \mathsf{bu}_{p,v})$ which is guaranteed since the buffer of t has a pending write on p not generated by a_1. Also notice that if $a_1 = \mathsf{rd}^{\circ}_{q,w}$ (where possibly $p = q$) then by Lemma 34 that there must be a preceding barrier b, which by the hypothesis $\mathsf{WB}(\gamma)$ cannot be a $\mathtt{wr}$. From the construction of the proof of Lemma 34 we know that the barrier b ($= \mathtt{ww}$) is ordered before $\mathsf{rd}^{\circ}_{q,w}$, so we can apply Corollary 36 which implies that there is a $\mathsf{wr}_{q,w}$ and a following $\mathsf{rd}_{q,v'}$ action in γ_1'; a contradiction to the semantics of buffers, provided by hypothesis $\mathsf{WB}(\gamma)$, since the write of p is pending and there is an intermediate $\mathtt{ww}$ barrier. Hence $a_1 \neq \mathsf{rd}^{\circ}_{q,w}$. This guarantees that the permutation of a_1 after the $\mathsf{bu}_{p,v}$ action preserves $\mathsf{WB}(\gamma')$.
- Suppose now that $\gamma_1'' = \xrightarrow[t',o_2]{a_2} \cdot \bar{\gamma_1}$. Consider the following cases:
  - if $t = t'$ we can verify once more that $a_1 \neq \mathsf{rd}^{\circ}_{q,w}$ since $[\gamma_0|_t, (\mathsf{wr}_{p,v}, o)] \prec_{\gamma|_t} [\gamma_0 \cdot \xrightarrow[o]{\mathsf{wr}_{p,v}} \cdot \gamma_1', (a_1, o_1)]$ in conjunction with Lemma 34 and Corollary 36 would violate the semantics of buffers granted by $\mathsf{WB}(\gamma)$.
    By cases on a_1. Suppose that $a_1 = \mathsf{rd}_{q,w}$ then we have from the semantics, that $q \neq p$ (no read can happen with pending writes). By Lemma 34 there must be a preceding barrier b and a following action that is dependent on b. Of course b cannot be a $\mathtt{wr}$ since it would violate the semantics of buffers. So it must be a $\mathtt{ww}$ and there must be an intermediate write on reference q that conflicts with the read $\mathsf{rd}_{q,w}$, as per Corollary 36. Thus, the preceding write on p should be committed before the one for q and there could not be a $\mathsf{rd}_{q,w}$ action. Hence $a_1 \neq \mathsf{rd}_{q,w}$. Otherwise suppose

that $a_1 = \mathsf{wr}_{q,w}$. If $q = p$ we have from the semantics of buffers that a_1 is not committed in σ_1'' and thus it is trivial to see that can be reordered after the buffer update (using Lemma 26, Lemma 27 and Lemma 23). So, in particular can be reordered with a_2. If $p \neq q$ then there must be an intermediate `ww`, else we would have no ordering (again as per Lemma 34 and Lemma 37); thus we know that the write is not committed in σ_1' (otherwise we would have a violation to the semantics of buffers) and we can permute a_1 after the buffer update, thus in particular after a_2. We observe then that the resulting computation γ' satisfies $\mathsf{WB}(\gamma')$ and we can use the induction hypothesis to conclude.

- if $t \neq t'$ we can trivially reorder a_1 (recall from the previous case that $a_1 \notin \{\mathsf{rd}_{q,w}, \mathsf{rd}^{\circ}_{q,w}\}$) with a_2 by Lemma 27 and conclude by the induction hypothesis. Notice that the permutation trivially preserves $\mathsf{WB}(\gamma')$.

□

It should be easy to see that similar (but simpler) results can be derived in the case where the first late-commit is a $\overline{\texttt{ww}}$ or $\overline{\texttt{wr}}$ action. What is important to observe here is that from the proof we know that only $\mathsf{wr}_{p,w}$, `ww` or `wr` actions need to be permuted. Read actions need never be reordered. We will use this observation in the sequel.

The following corollary states that we can always find a speculation where writes are immediately followed by their corresponding buffer updates. Clearly the same holds for barrier actions as shown in the subsequent corollaries. This property is the core of the proof establishing that the semantics of speculations can simulate the one of write-buffers.

Corollary 41 (Matching Write Update). *Let $\gamma : C \xrightarrow{*} C'$ be a RMNF computation such that $\mathsf{WB}(\gamma)$ and $\gamma = \sigma_0 \cdot \xrightarrow[t,o]{\mathsf{wr}_{p,v}} \cdot\ \sigma_1 \cdot \xrightarrow[t,\varepsilon]{\mathsf{bu}_{p,v}} \cdot\ \sigma_2$ with $\sigma_0 \cdot \xrightarrow[t,o]{\mathsf{wr}_{p,v}} \cdot\ \sigma_1$ the longest late-commit free prefix of γ, and with $[\sigma_0|_t, (\mathsf{wr}_{p,v}, o)]$ the first uncommitted write on reference p of thread t in γ. Then there exists o', σ_1' and σ_1'' and $\gamma' : C \xrightarrow{*} C'$ such that*

$$\gamma' = \sigma_0 \ \cdot\ \sigma_1' \ \cdot\ \xrightarrow[t,o']{\mathsf{wr}_{p,v}} \ \cdot\ \xrightarrow[t,\varepsilon]{\mathsf{bu}_{p,v}} \cdot\ \sigma_1'' \ \cdot\ \sigma_2$$

and for all $t' \neq t$ we have $\gamma|'_t = \gamma'|'_t$ and $\gamma'|_t \propto^{MG} \gamma|_t$. Moreover $\mathsf{WB}(\gamma')$ holds.

Proof. Simple induction on σ_1 repeatedly applying Lemma 40, Lemma 26 and Lemma 27. □

Corollary 42 (Matching Barrier Update). *Let $\gamma : C \xrightarrow{*} C'$ be a RMNF computation such that $\mathsf{WB}(\gamma)$ and $\gamma = \sigma_0 \cdot \xrightarrow[t,o]{b} \cdot\ \sigma_1 \cdot \xrightarrow[t,\varepsilon]{\overline{b}} \cdot\ \sigma_2$ with $b \in \mathcal{S}ync$ and $\sigma_0 \cdot \xrightarrow[t,o]{b} \cdot\ \sigma_1$ the longest late-commit free prefix of γ, and with $[\sigma_0|_t, (b, o)]$ the first uncommitted b barrier of thread t in γ. Then there exists o', σ_1' and σ_1''*

and $\gamma' : C \xrightarrow{*} C'$ *such that*

$$\gamma' = \sigma_0 \cdot \sigma'_1 \cdot \xrightarrow[t,o']{b} \cdot \xrightarrow[t,\varepsilon]{\bar{b}} \cdot \sigma''_1 \cdot \sigma_2$$

and for all $t' \neq t$ *we have* $\gamma|'_t = \gamma'|'_t$ *and* $\gamma'|_t \propto^{MG} \gamma|_t$. *Moreover* $\mathsf{WB}(\gamma')$ *holds.*

Proof. Same as for the previous lemma. □

To establish a relation between computations in the formalization with write-buffers and the formalization with speculations we need to identify which are the computations of the merge-calculus that correspond to computations of the speculative semantics. For that purpose we will define *quasi-speculative* computations.

Definition 43 (Quasi-speculative Computation). *A computation* $\gamma : C \xrightarrow{*} C'$ *of the merge-calculus is called* quasi-speculative *if every write is immediately followed by its corresponding buffer update, and every barrier action is followed immediately by its corresponding barrier commit.*

Lemma 44. *Given* $\gamma : C \xrightarrow{*} C'$ *a computation of the merge-calculus satisfying* $\mathsf{WB}(\gamma)$ *there exists a quasi-speculative computation* γ' *of the merge-calculus such that for every thread* t *we have* $\gamma'|_t \propto^{MG} \gamma|_t$.

Proof. The proof proceeds by induction on the number of buffer update (including barrier commit) actions present in γ and orders writes, or barriers to match their respective buffer-update action as stated by corollaries 41 and 42. □

In a quasi-speculative computation of the merge-calculus we still have the buffer updates and the commits of barrier symbols in the buffers. To obtain a truly-speculative computation we need to erase these actions from the computation. Let us denote by $\lfloor\gamma\rfloor$ the computation that results from erasing all commit actions from the merge-calculus computation γ.

Theorem 15 (Write Buffering ⇒ Speculations)**.** *Given a computation* $\gamma : C \xrightarrow{*} C'$ *of the formalization of PSO with write-buffers (as defined in Figure 4) there exists a quasi-speculative computation* $\gamma' : C \xrightarrow{*} C'$ *such that for every thread* t *it holds* $\gamma'|_t \propto^{MG} \gamma|_t$. *Then* $\lfloor\gamma'\rfloor$ *is a* $\mathcal{D}^{PSO}$*-valid speculative computation.*

Proof. Given the computation γ we have from remarks 13 and 13 that γ is a computation of the merge-calculus and in particular we have $\mathsf{WB}(\gamma)$. We can therefore apply Lemma 44 to obtain $\gamma' : C \xrightarrow{*} C'$ a quasi-speculative computation. It is not hard to see from the construction of γ' that since the execution γ satisfies $\mathsf{WB}(\gamma)$ the final computation $\lfloor\gamma'\rfloor$ is a $\mathcal{D}^{PSO}$-valid speculative computation. □

Appendix B.5 From Speculations to Write-Buffers

Let us see now how a computation of the speculative formalization of PSO can be turned into one of the formalization with write buffers by reordering speculatively performed actions to the position where they become "normal". In essence, since we consider only $\mathcal{D}^{PSO}$-valid speculations, we know that for every thread projection $\gamma|_t$, for an hypothetical speculation γ, there exists a *normal* speculation that is a $\mathcal{D}^{PSO}$-reordering of $\gamma|_t$, and let us denote such normal computation $\gamma_{[t]}$. We prove here that we can always find a computation γ' of the merge-calculus such that coincides with γ, the initial and final states are the same, and for every thread $\gamma'|_t = \gamma_{[t]}$. Let us now proceed with the proof.

Given a valid speculative computation γ we can trivially obtain a quasi-speculative computation $\lceil\gamma\rceil$ of the merge-calculus where all writes and barrier instructions are immediately committed.

Remark 45. Given a valid computation $\gamma : C \xrightarrow{*} C'$ of the speculative calculus there is a quasi-speculative computation $\lceil\gamma\rceil : C \xrightarrow{*} C'$ of the merge-calculus such that for all t we have $\gamma|_t = \gamma'|_t$.

We now prove that buffer commit actions can be reordered w.r.t. every action other than a read or a buffer commit action of the same thread to reach the same final configuration.

Lemma 46. *Suppose that we have* $C \xrightarrow[t,\varepsilon]{a_0} C_0 \xrightarrow[t,o_1]{a_1} C'$ *and* $a_0 \in \{\mathsf{bu}_{p,v}, \overline{\mathtt{ww}}, \overline{\mathtt{wr}} \mid p \in \mathcal{R}ef, v \in \mathcal{V}al\}$ *and* $a_1 \notin \{\mathsf{rd}_{p,v}, \mathsf{bu}_{p,v}, \overline{\mathtt{ww}}, \overline{\mathtt{wr}} \mid p \in \mathcal{R}ef, v \in \mathcal{V}al\}$*, then there exists* C_1 *such that* $C \xrightarrow[t,o_1]{a_1} C_1 \xrightarrow[t,\varepsilon]{a_0} C'$.

Proof. The proof is trivial by case analysis. Notice that no a_1 action other than a write modifies or depends on the buffers or memory. In the case of a write action, it simply puts its contents at the end of the buffer which is independent of any previous buffer update. □

If an action does not conflict with preceding actions of different threads, then this action can be moved backwards in the computation obtaining an equivalent computation (in the sense that individual speculations are preserved and the initial and final configurations are the same).

Lemma 47. *Let* $\gamma = C \xrightarrow{*} C'$ *be a computation of the merge-calculus and let* $\gamma = \gamma_0 \cdot \xrightarrow[t,o_0]{a_0} \cdot \gamma_1 \cdot \xrightarrow[t,o_1]{a_1} \cdot \gamma_2$ *with* $a_1 \notin \{\mathsf{rd}_{p,v}, \mathsf{bu}_{p,v}, \overline{\mathtt{wr}}, \overline{\mathtt{ww}} \mid p \in \mathcal{R}ef, v \in \mathcal{V}al\}$ *and* $\gamma_1|_t = \epsilon$*. Then there exists* $\gamma' = C \xrightarrow{*} C'$ *and* γ'_1 *such that for all* t *we have* $\gamma|_t = \gamma'|_t$ *and* $\gamma' = \gamma_0 \cdot \xrightarrow[t,o_0]{a_0} \cdot \xrightarrow[t,o_1]{a_1} \cdot \gamma'_1 \cdot \gamma_2$.

Proof. We proceed by induction on the length of γ_1. Clearly if $\gamma_1 = \varepsilon$ we have the conclusion. If $\gamma_1 = \overline{\gamma_1} \cdot \xrightarrow[t_2,o_2]{a_2}$ we have the following cases:

- $t = t_2$ and therefore we know that $a_2 \in \{\mathsf{bu}_p, \overline{\mathtt{wr}}, \overline{\mathtt{ww}} \mid p \in \mathcal{R}ef\}$ by $\gamma_1|_t = \varepsilon$. We can simply apply the previous lemma (46) and conclude by the induction hypothesis.

– $t \neq t_2$ and since $a_1 \notin \{\mathsf{rd}_{p,v}, \mathsf{bu}_{p,v}, \overline{\mathtt{ww}}, \overline{\mathtt{wr}} \mid p \in \mathcal{R}ef, v \in \mathcal{V}al\}$ we can directly apply Lemma 27 and the induction hypothesis to conclude.

□

And similarly to the previous result, if an action does not conflict with subsequent actions of different threads in the computation, we can push it forward to obtain an equivalent computation.

Lemma 48. *Let $\gamma : C \xrightarrow{*} C'$ be a merge-calculus computation such that $\gamma = \gamma_0 \cdot \xrightarrow[t,o]{a} \cdot \gamma_1 \cdot \gamma_2$ with $a \notin \{\mathsf{wr}_{p,v}, \mathsf{rd}_{p,v}, \mathsf{bu}_{pv}, \overline{\mathtt{ww}}, \overline{\mathtt{wr}} \mid p \in \mathcal{R}ef, v \in \mathcal{V}al\}$ and $\gamma_1|_t = \epsilon$. Then there are γ' and γ'_1 such that $\gamma' = \gamma_0 \cdot \gamma'_1 \cdot \xrightarrow[t,o]{a} \cdot \gamma_2$ with $\gamma' : C \xrightarrow{*} C'$ and for all $t \in \mathcal{T}id$ we have $\gamma|_t = \gamma'|_t$.*

Proof. The proof is obvious since a does not modify or inspect the buffers or the memory. □

Lemma 49. *Let $\gamma : C \xrightarrow{*} C'$ be a merge-calculus computation such that every thread projection is* valid*, i.e. for all $t \in \mathcal{T}id$ then $\gamma|_t \propto^{MG} \gamma_{[t]}$ with $\gamma_{[t]}$ a normal speculation. Suppose as well that $\gamma|_t \propto^{MG} \gamma'_t \propto^{MG} \gamma_{[t]}$ with $\gamma|_t$ reaching γ'_t by a single reordering. Namely $\gamma = \gamma_0 \cdot \xrightarrow[t,o_0]{a_0} \cdot \gamma_1 \cdot \xrightarrow[t,o_1]{a_1} \cdot \gamma_2$ with $\gamma_1|_t = \epsilon$ and $\gamma'_t = \gamma_0|_t \cdot \xrightarrow[o'_1]{a_1} \cdot \xrightarrow[o'_0]{a_0} \cdot \gamma_2|_t$. Then there exists γ'_1, γ''_1 and $\gamma' : C \xrightarrow{*} C'$ such that $\gamma' = \gamma_0 \cdot \gamma'_1 \cdot \xrightarrow[t,o'_1]{a_1} \cdot \xrightarrow[t,o'_0]{a_0} \cdot \gamma''_1 \cdot \gamma_2$. Moreover for all $t' \in \mathcal{T}id$ with $t' \neq t$ we have $\gamma'|_t = \gamma|_t$.*

Proof. Let us proceed by cases on a_1:

– If $a_1 \in \{\mathsf{rd}_{p,v}, \mathsf{rd}^{\circ}_{p,v} \mid p \in \mathcal{R}ef, v \in \mathcal{V}al\}$ we have from the fact that $\gamma|_t \propto^{MG} \gamma'_t$ that $\neg(a_1 \mathcal{D}^{PSO} a_0)$ which implies that $a_0 \notin \{\mathsf{rd}_{q,w}, \mathsf{rd}^{\circ}_{q,w}, \mathsf{wr}_{q,w} \mid q \in \mathcal{R}ef, w \in \mathcal{V}al\}$ and thus we can apply Lemma 48 and Lemma 26 to conclude.
– If $a_1 = \mathsf{wr}_{p,v}$ we can apply Lemma 47 and Lemma 26 to get the desired reordering.
– If If $a_1 = \{\mathtt{ww}, \mathtt{wr}\}$ we proceed as in the previous case.
– All the remaining cases are trivial.

□

One remark from the proof of this lemma is that it requires to reorder only memory-model-related actions (that is reads, writes or memory barriers) forward in the computation (in particular in the speculation). One can observe that the actions that need to be reordered are write actions and barrier actions, where read actions can be regarded as remaining at their place. Indeed, we see in the proof that if a_1 is a read action the a_0 action is moved to a later stage in the computation (but in this case a_0 is not a memory model related action), and in the cases where a_1 is a write or a barrier, it is this action that is moved to the front of the computation.

The following lemma states that merge-calculus computations corresponding to $\mathcal{D}^{PSO}$-valid speculative computations can be reordered to reach a computation of the semantics with write-buffers.

Lemma 50. *Let $\lceil\gamma\rceil : C \xrightarrow{*} C'$ be a quasi-speculative computation of the merge-calculus corresponding to a $\mathcal{D}^{PSO}$-valid speculative computation γ of the formalization of PSO by means of speculations, and let $\gamma' : C \xrightarrow{*} C'$ be a merge-calculus computation such that for all $t \in \mathcal{T}id$, $\gamma|_t \propto^{MG} \gamma'|_t$. Then there is $\gamma'' : C \xrightarrow{*} C'$ a purely buffered computation of the merge-calculus such that for all $t \in \mathcal{T}id$, $\gamma'|_t \propto^{MG} \gamma''|_t$.*

Proof. The proof is by induction on the summation of reorderings needed to reach $\gamma_{[t]}$ from $\gamma'|_t$ for every t. In the base case all projections are normal, and thus the computation corresponds to a computation of the calculus with write buffers. We simply apply the previous lemma (49) in the induction case and conclude by the induction hypothesis. Notice as well that the validity condition guarantees that $\mathsf{rd}^{\circ}_{p,v}$ actions are actually allowed and moreover obtain the correct value from the buffers. □

And we can finally prove that $\mathcal{D}^{PSO}$-valid computations in the speculative semantics of PSO correspond to computations of the semantics of PSO with write-buffers.

Theorem 16 (Speculations ⇒ Write Buffers)**.** *Given $\gamma : C \xrightarrow{*} C'$ a $\mathcal{D}^{PSO}$-valid speculative computation of the formalization of PSO with speculations. There exists a purely buffered computation of the merge-calculus $\gamma'' : C \xrightarrow{*} C$ such that for all $t \in \mathcal{T}id$ then $\gamma|_t \propto^{MG} \gamma''|_t = \gamma_{[t]}$. And in particular γ'' is a computation of the calculus with buffers.*

Proof. The proof is simple consequence of Remark 45 and Lemma 50. □