x86-TSO

The HOL Specification

Scott Owens   Susmit Sarkar   Peter Sewell

University of Cambridge

March 25, 2009

Full Contents

Introduction

This document is an automatically typeset version of the HOL definition of our x86-TSO model.

Part I

axiomatic_memory_model

type_abbrev Ximm : word32

type_abbrev proc : num

$iiid = \langle\!|$ proc : proc;
$\quad poi$: num$|\!\rangle$

type_abbrev address : Ximm

type_abbrev value : Ximm

type_abbrev eiid : num

type_abbrev reln : $'a \# 'a \rightarrow$ bool

dirn $= R \mid \mathrm{W}$

location $=$ $\textsc{Location\_reg}$ **of** proc $'reg$
$\quad \mid \textsc{Location\_mem}$ **of** address

barrier $= \textsc{Lfence} \mid \textsc{Sfence} \mid \textsc{Mfence}$

$action = \textsc{Access}$ **of** dirn ($'reg$ location) value $\mid \textsc{Barrier}$ **of** barrier

event $= \langle\!|$ eiid : eiid;
$\quad iiid : iiid$;
$\quad action : ('reg\ action)|\!\rangle$

event_structure $= \langle\!|$ $procs$: proc set;
$\quad events$: ($'reg$ event)set;
$\quad intra\_causality$: ($'reg$ event)reln;
$\quad atomicity$: ($'reg$ event)set set$|\!\rangle$

$\mathrm{is\_mem\_access}\ e = \exists d\ a\ v. e.action = \textsc{Access}\ d\ (\textsc{Location\_mem}\ a) v$

$\mathrm{writes}\ E = \{e \mid e \in E.events \wedge \exists l\ v. e.action = \textsc{Access}\ \mathrm{W}\ l\ v\}$

$\mathrm{reads}\ E = \{e \mid e \in E.events \wedge \exists l\ v. e.action = \textsc{Access}\ R\ l\ v\}$

$\mathrm{fences}\ E = \{e \mid e \in E.events \wedge (\exists f. e.action = \textsc{Barrier}\ f)\}$

mfences $E = \{e \mid e \in E.events \wedge (e.action = \textsc{Barrier Mfence})\}$

mem_writes $E = \{e \mid e \in E.events \wedge \exists a\ v.e.action = \textsc{Access}\ \mathrm{W}\ (\textsc{Location\_mem}\ a) v\}$

mem_reads $E = \{e \mid e \in E.events \wedge \exists a\ v.e.action = \textsc{Access}\ R\ (\textsc{Location\_mem}\ a) v\}$

reg_writes $E = \{e \mid e \in E.events \wedge \exists p\ r\ v.e.action = \textsc{Access}\ \mathrm{W}\ (\textsc{Location\_reg}\ p\ r) v\}$

reg_reads $E = \{e \mid e \in E.events \wedge \exists p\ r\ v.e.action = \textsc{Access}\ R\ (\textsc{Location\_reg}\ p\ r) v\}$

mem_accesses $E = \{e \mid e \in E.events \wedge (\exists d\ a\ v.e.action = \textsc{Access}\ d\ (\textsc{Location\_mem}\ a) v)\}$

reg_accesses $E = \{e \mid e \in E.events \wedge \exists d\ p\ r\ v.e.action = \textsc{Access}\ d\ (\textsc{Location\_reg}\ p\ r) v\}$

```
loc  $e$ =
  case  $e.action$  of
     Access $d\ l\ v \rightarrow$ Some  $l$
  || Barrier $f \rightarrow$ None
```

```
value_of  $e$ =
  case  $e.action$  of
     Access $d\ l\ v \rightarrow$ Some  $v$
  || Barrier $f \rightarrow$ None
```

proc $e = e.iiid.\mathsf{proc}$

po_strict $E =$
$\{(e_1, e_2) \mid (e_1.iiid.\mathsf{proc} = e_2.iiid.\mathsf{proc}) \wedge e_1.iiid.poi < e_2.iiid.poi \wedge$
$e_1 \in E.events \wedge e_2 \in E.events\}$

po_iico $E =$ po_strict $E \cup E.intra\_causality$

well_formed_event_structure $E =$
(* The set of events is at most countable *)
countable $E.events \wedge$

(* there are only a finite number of processors *)
(finite $E.procs) \wedge$

(* all events are from one of those processors *)
$(\forall e \in (E.events).\mathsf{proc}\ e \in E.procs) \wedge$

(* the eiid and iiid of an event (together) identify it uniquely *)

$(\forall e_1\ e_2 \in (E.events).(e_1.\mathsf{eiid} = e_2.\mathsf{eiid}) \wedge (e_1.iiid = e_2.iiid) \implies (e_1 = e_2)) \wedge$

(* intra-instruction causality is a partial order over the events *)
$\text{partial\_order}\ (E.intra\_causality) E.events \wedge$

(* ...and moreover, is *intra*-instruction *)
$(\forall (e_1, e_2) \in (E.intra\_causality).(e_1.iiid = e_2.iiid)) \wedge$

(* the atomicity data is a partial equivalence relation: the atomic sets of events are disjoint *)
$\text{per}\ \ E.events\ E.atomicity \wedge$

(* atomic sets are *intra* instruction *)
$(\forall es \in (E.atomicity).\forall e_1\ e_2 \in es.(e_1.iiid = e_2.iiid)) \wedge$

(* accesses to a register on a processor can only be by that processor *)
$(\forall e \in (E.events).\forall p\ r.(\text{loc}\ \ e = \textsc{Some}\ \ (\textsc{Location\_reg}\ p\ r)) \implies (p = \mathsf{proc}\ e)) \wedge$

(* An event never comes after an infinite number of other events in program order *)
$\text{finite\_prefixes}\ (\text{po\_iico}\ \ E) E.events \wedge$

(* The additional properties below hold, for the ISA fragment dealt with in [SSFN+09], and were useful for the metatheory there, but seem less essential than those above. *)

(* there is no intra-causality edge *from* a memory write *)
$(\forall (e_1, e_2) \in (E.intra\_causality).e_1 \neq e_2 \implies e_1 \notin \text{mem\_writes}\ \ E) \wedge$

(* if an instruction two events on a location and one is a write, then there must be an intra-causality edge between them. In other words, there cannot be a local race within an instruction *)

```
$(\forall (e_1 \in \text{writes}\ \ E) e_2.$
   $(e_1 \neq e_2) \wedge$
   $(e_2 \in \text{writes}\ \ E \vee e_2 \in \text{reads}\ \ E) \wedge$
   $(e_1.iiid = e_2.iiid) \wedge$
   $(\text{loc}\ \ e_1 = \text{loc}\ \ e_2)$
    $\implies$
   $(e_1, e_2) \in E.intra\_causality \vee$
   $(e_2, e_1) \in E.intra\_causality) \wedge$
```

(* each atomic set includes all the events of its instruction *)
$(\forall es \in (E.atomicity).\forall e_1 \in es.\forall e_2 \in (E.events).(e_1.iiid = e_2.iiid) \implies e_2 \in es) \wedge$

(* all locked instructions include at least one memory read *)
$(\forall es \in (E.atomicity).\exists e \in es.e \in \text{mem\_reads}\ \ E)$

$\mathsf{execution\_witness} =$
$\langle\!|\ memory\_order : ('reg\ \mathsf{event})\mathsf{reln};$
$\quad rfmap : ('reg\ \mathsf{event})\mathsf{reln};$
$\quad initial\_state : ('reg\ \mathsf{location} \to \mathsf{value}\ \ \mathsf{option})|\!\rangle$

$\text{previous\_writes}\ \ E\ er\ order =$

```
  $\{ew' \mid ew' \in \text{writes } E \wedge (ew', er) \in \textit{order} \wedge (\text{loc } ew' = \text{loc } er)\}$


$\text{check\_rfmap\_written } E\ X =$
  $\forall (ew, er) \in (X.\textit{rfmap}).$
    **if** $ew \in \text{mem\_accesses } E$ **then**
      $ew \in \text{maximal\_elements } (\text{previous\_writes } E\ er\ X.\textit{memory\_order} \cup$
                                      $\text{previous\_writes } E\ er\ (\text{po\_iico } E))$
                                   $X.\textit{memory\_order}$
    **else** (* ew IN reg_accesses E *)
      $ew \in \text{maximal\_elements } (\text{previous\_writes } E\ er\ (\text{po\_iico } E))(\text{po\_iico } E)$


$\text{check\_rfmap\_initial } E\ X =$
  $\forall er \in (\text{reads } E \setminus \text{range } X.\textit{rfmap}).$
    $(\exists l.(\text{loc } er = \text{SOME } l) \wedge (\text{value\_of } er = X.\textit{initial\_state } l)) \wedge$
    $(\text{previous\_writes } E\ er\ X.\textit{memory\_order} \cup$
      $\text{previous\_writes } E\ er\ (\text{po\_iico } E) = \{\})$


$\text{reads\_from\_map\_candidates } E\ \textit{rfmap} =$
  $\forall (ew, er) \in \textit{rfmap}.(er \in \text{reads } E) \wedge (ew \in \text{writes } E) \wedge$
                        $(\text{loc } ew = \text{loc } er) \wedge (\text{value\_of } ew = \text{value\_of } er)$


$\text{valid\_execution } E\ X =$
  $\text{partial\_order } X.\textit{memory\_order } (\text{mem\_accesses } E) \wedge$
  $\text{linear\_order } (X.\textit{memory\_order}|_{(\text{mem\_writes } E)})(\text{mem\_writes } E) \wedge$
  $\text{finite\_prefixes } X.\textit{memory\_order } (\text{mem\_accesses } E) \wedge$
  $(\forall ew \in (\text{mem\_writes } E).$
    $\text{finite}\{er \mid er \in E.\textit{events} \wedge (\text{loc } er = \text{loc } ew) \wedge$
                        $(er, ew) \notin X.\textit{memory\_order} \wedge (ew, er) \notin X.\textit{memory\_order}\}) \wedge$
  $(\forall er \in (\text{mem\_reads } E).\forall e \in (\text{mem\_accesses } E).(er, e) \in \text{po\_iico } E \implies (er, e) \in X.\textit{memory\_order}) \wedge$
  $(\forall ew_1\ ew_2 \in (\text{mem\_writes } E).(ew_1, ew_2) \in \text{po\_iico } E \implies (ew_1, ew_2) \in X.\textit{memory\_order}) \wedge$
  $(\forall ew \in (\text{mem\_writes } E).\forall er \in (\text{mem\_reads } E).\forall ef \in (\text{mfences } E).$
    $(ew, ef) \in \text{po\_iico } E \wedge (ef, er) \in \text{po\_iico } E \implies (ew, er) \in X.\textit{memory\_order}) \wedge$
  $(\forall e_1\ e_2 \in (\text{mem\_accesses } E).\forall es \in (E.\textit{atomicity}).$
    $(e_1 \in es \vee e_2 \in es) \wedge (e_1, e_2) \in \text{po\_iico } E$
     $\implies$
    $(e_1, e_2) \in X.\textit{memory\_order}) \wedge$
  $(\forall es \in (E.\textit{atomicity}).\forall e \in (\text{mem\_accesses } E \setminus es).$
    $(\forall e' \in (es \cap \text{mem\_accesses } E).(e, e') \in X.\textit{memory\_order}) \vee$
    $(\forall e' \in (es \cap \text{mem\_accesses } E).(e', e) \in X.\textit{memory\_order})) \wedge$
  $X.\textit{rfmap} \in \text{reads\_from\_map\_candidates } E \wedge$
  $\text{check\_rfmap\_written } E\ X \wedge$
  $\text{check\_rfmap\_initial } E\ X$


$\text{linear\_valid\_execution } E\ X =$
  $\text{linear\_order } X.\textit{memory\_order } (\text{mem\_accesses } E) \wedge$
  $\text{finite\_prefixes } X.\textit{memory\_order } (\text{mem\_accesses } E) \wedge$
  $(\forall er \in (\text{mem\_reads}\ E).\forall e \in (\text{mem\_accesses}\ E).(er, e) \in \text{po\_iico}\ E \implies (er, e) \in X.\mathit{memory\_order}) \wedge$
  $(\forall ew_1\ ew_2 \in (\text{mem\_writes}\ E).(ew_1, ew_2) \in \text{po\_iico}\ E \implies (ew_1, ew_2) \in X.\mathit{memory\_order}) \wedge$
  $(\forall ew \in (\text{mem\_writes}\ E).\forall er \in (\text{mem\_reads}\ E).\forall ef \in (\text{mfences}\ E).$
    $(ew, ef) \in \text{po\_iico}\ E \wedge (ef, er) \in \text{po\_iico}\ E \implies (ew, er) \in X.\mathit{memory\_order}) \wedge$
  $(\forall e_1\ e_2 \in (\text{mem\_accesses}\ E).\forall es \in (E.\mathit{atomicity}).$
    $(e_1 \in es \vee e_2 \in es) \wedge (e_1, e_2) \in \text{po\_iico}\ E$
      $\implies$
    $(e_1, e_2) \in X.\mathit{memory\_order}) \wedge$
  $(\forall es \in (E.\mathit{atomicity}).\forall e \in (\text{mem\_accesses}\ E \setminus es).$
    $(\forall e' \in (es \cap \text{mem\_accesses}\ E).(e, e') \in X.\mathit{memory\_order}) \vee$
    $(\forall e' \in (es \cap \text{mem\_accesses}\ E).(e', e) \in X.\mathit{memory\_order})) \wedge$
  $X.\mathit{rfmap} \in \text{reads\_from\_map\_candidates}\ E \wedge$
  $\text{check\_rfmap\_written}\ E\ X \wedge$
  $\text{check\_rfmap\_initial}\ E\ X$
```

```
$\text{max\_state\_updates}\ E\ X\ l =$
  $\{\text{value\_of}\ ew \mid ew \in \text{maximal\_elements}$
                              $\{ew' \mid ew' \in \text{writes}\ E \wedge (\text{loc}\ ew' = \textsc{Some}\ l)\}$
                              $(\textbf{case}\ l\ \textbf{of}$
                                   $\textsc{Location\_mem}\ a \to X.\mathit{memory\_order}$
                                 $\parallel \textsc{Location\_reg}\ p\ r \to \text{po\_iico}\ E)\}$
```

```
$(\text{check\_final\_state}\ E\ X\ \textsc{None} =$
  $\neg(\text{finite}\ E.\mathit{events})) \wedge$
$(\text{check\_final\_state}\ E\ X\ (\textsc{Some}\ \mathit{final\_state}) =$
  $\text{finite}\ E.\mathit{events} \wedge$
  $(\forall l.$
    $\textbf{if}\ (\text{max\_state\_updates}\ E\ X\ l) = \{\}\ \textbf{then}$
      $\mathit{final\_state}\ l = X.\mathit{initial\_state}\ l$
    $\textbf{else}$
      $\mathit{final\_state}\ l \in \text{max\_state\_updates}\ E\ X\ l))$
```

Part II

typesetting

$<_{order} = order$

$e <_{order} e' = (e, e') \in \ order$

$e \not<_{order} e' = (e, e') \notin \ order$

previous_writes $E\ er\ order =$
$\quad \{ew' \mid ew' \ \in \ \text{writes}\ E \wedge ew' <_{order} er \wedge (\text{loc}\ \ ew' = \text{loc}\ \ er)\}$

```
check_rfmap_written  E X =
  ∀(ew, er) ∈ (X.rfmap).
    if  ew ∈ mem_accesses E  then
      ew ∈ maximal_elements (previous_writes E er (<_{X.memory_order}) ∪
                                    previous_writes E er (<_{(po_iico E)}))
                               (<_{X.memory_order})
    else (* ew IN reg_accesses E *)
      ew ∈ maximal_elements (previous_writes E er (<_{(po_iico E)}))(<_{(po_iico E)})
```

check_rfmap_initial $E\ X =$
$\quad \forall er \in (\text{reads}\ \ E\ \setminus\ \text{range}\ \ X.rfmap).$
$\quad\quad (\exists l.(\text{loc}\ \ er = \textsc{Some}\ \ l) \wedge (\text{value\_of}\ \ er = X.initial\_state\ l)) \wedge$
$\quad\quad (\text{previous\_writes}\ \ E\ er\ (<_{X.memory\_order}) \cup$
$\quad\quad\quad \text{previous\_writes}\ \ E\ er\ (<_{(\text{po\_iico}\ E)}) = \{\})$

ve1 $E\ X =$
partial_order $(<_{X.memory\_order})(\text{mem\_accesses}\ \ E)$

ve2 $E\ X =$
linear_order $((<_{X.memory\_order})|_{(\text{mem\_writes}\ E)})(\text{mem\_writes}\ \ E)$

ve3 $E\ X =$
finite_prefixes $(<_{X.memory\_order})(\text{mem\_accesses}\ \ E)$

ve4 $E\ X =$
$\forall ew \in (\text{mem\_writes}\ \ E).$
$\quad \text{finite}\{er \mid er \ \in \ E.events \wedge (\text{loc}\ \ er = \text{loc}\ \ ew) \wedge$
$\quad\quad\quad er \not<_{X.memory\_order} ew \wedge ew \not<_{X.memory\_order} er\}$

ve5 $E\ X =$
$\forall er \in (\text{mem\_reads}\ \ E).\forall e \in (\text{mem\_accesses}\ \ E).$
$\quad er <_{(\text{po\_iico}\ E)} e \implies er <_{X.memory\_order} e$

ve6 $E\ X =$
$\forall ew_1\ ew_2 \in (\mathrm{mem\_writes}\ E).$
$\quad ew_1 <_{(\mathrm{po\_iico}\ E)} ew_2 \implies ew_1 <_{X.memory\_order} ew_2$

ve7 $E\ X =$
$\forall ew \in (\mathrm{mem\_writes}\ E).\forall er \in (\mathrm{mem\_reads}\ E).\forall ef \in (\mathrm{mfences}\ E).$
$\quad (ew <_{(\mathrm{po\_iico}\ E)} ef \wedge ef <_{(\mathrm{po\_iico}\ E)} er) \implies ew <_{X.memory\_order} er$

ve8 $E\ X =$
$\forall e_1\ e_2 \in (\mathrm{mem\_accesses}\ E).\forall es \in (E.atomicity).$
$\quad ((e_1 \in es \vee e_2 \in es) \wedge e_1 <_{(\mathrm{po\_iico}\ E)} e_2) \implies e_1 <_{X.memory\_order} e_2$

ve9 $E\ X =$
$\forall es \in (E.atomicity).\forall e \in (\mathrm{mem\_accesses}\ E \setminus es).$
$\quad (\forall e' \in (es \cap \mathrm{mem\_accesses}\ E).e <_{X.memory\_order} e') \vee$
$\quad (\forall e' \in (es \cap \mathrm{mem\_accesses}\ E).e' <_{X.memory\_order} e)$

ve10 $E\ X = \mathrm{reads\_from\_map\_candidates}\ E\ X.rfmap$

Part III

moretypesetting

$f \oplus (k \mapsto v) = (k \mapsto v) f$

Read from memory

$$\frac{\text{not\_blocked}\ \ s\ p \wedge (s.M\ \ a = \textsc{Some}\ \ v) \wedge \text{no\_pending}\ (s.B\ \ p) a}{s\ \ \xrightarrow{\textsc{Evt}\ p\ (\textsc{Access}\ R\ (\textsc{Location\_mem}\ a) v)}\ \ s}$$

Read from write buffer

$$\frac{\text{not\_blocked}\ \ s\ p \wedge (\exists b_1\ b_2.(s.B\ \ p = b_1 +\!\!+ [(a,v)] +\!\!+ b_2) \wedge \text{no\_pending}\ \ b_1\ \ a)}{s\ \ \xrightarrow{\textsc{Evt}\ p\ (\textsc{Access}\ R\ (\textsc{Location\_mem}\ a) v)}\ \ s}$$

Read from register

$$\frac{(s.R\ p\ r = \textsc{Some}\ \ v)}{s\ \ \xrightarrow{\textsc{Evt}\ p\ (\textsc{Access}\ R\ (\textsc{Location\_reg}\ p\ r) v)}\ \ s}$$

Write to write buffer

$$\frac{\mathbf{T}}{s\ \ \xrightarrow{\textsc{Evt}\ p\ (\textsc{Access}\ \mathrm{W}\ (\textsc{Location\_mem}\ a) v)}\ \ s\ \oplus \langle\!| B := s.B \oplus (p \mapsto [(a,v)] +\!\!+ (s.B\ \ p)) |\!\rangle}$$

Write from write buffer to memory

$$\frac{\text{not\_blocked}\ \ s\ p \wedge (s.B\ \ p = b +\!\!+ [(a,v)])}{s\ \ \xrightarrow{\textsc{Tau}}\ \ s\ \oplus \langle\!| M := s.M \oplus (a \mapsto \textsc{Some}\ \ v); B := s.B \oplus (p \mapsto b) |\!\rangle}$$

Write to register

$$\frac{\mathbf{T}}{s\ \ \xrightarrow{\textsc{Evt}\ p\ (\textsc{Access}\ \mathrm{W}\ (\textsc{Location\_reg}\ p\ r) v)}\ \ s\ \oplus \langle\!| R := s.R \oplus (p \mapsto ((s.R\ \ p) \oplus (r \mapsto \textsc{Some}\ \ v))) |\!\rangle}$$

Barrier

$$\frac{(b = \textsc{Mfence}) \implies (s.B\ \ p = [\,])}{s\ \ \xrightarrow{\textsc{Evt}\ p\ (\textsc{Barrier}\ b)}\ \ s}$$

Lock

$$\frac{(s.L = \textsc{None}) \wedge (s.B\ \ p = [\,])}{s\ \ \xrightarrow{\textsc{Lock}\ p}\ \ s\ \oplus \langle\!| L := \textsc{Some}\ \ p |\!\rangle}$$

Unlock

$$\frac{(s.L = \textsc{Some}\ \ p) \wedge (s.B\ \ p = [\,])}{s\ \ \xrightarrow{\textsc{Unlock}\ p}\ \ s\ \oplus \langle\!| L := \textsc{None} |\!\rangle}$$

Part IV

lts_memory_model

clause_name $x = \mathbf{T}$

machine_state $= \langle\!|\ R : \mathsf{proc} \to {}'reg \to \mathsf{value\ \ option};$ (* per-processor registers *)
$M : \mathsf{address} \to \mathsf{value\ \ option};$ (* main memory *)
$B : \mathsf{proc} \to (\mathsf{address\#value})\mathsf{list};$ (* per-processor write buffers *)
$L : \mathsf{proc\ \ option}$(* which processor holds the lock *)$|\!\rangle$

not_blocked $s\ p = (s.L = \textsc{None}) \lor (s.L = \textsc{Some}\ \ p)$

no_pending $b\ a = \neg(\exists v'.\, \mathrm{MEM}\ (a, v')b)$

label $= \textsc{Tau} \mid \textsc{Evt}\ \ \mathbf{of}\ \ \mathsf{proc}\ ({}'reg\ action) \mid \textsc{Lock}\ \ \mathbf{of}\ \ \mathsf{proc} \mid \textsc{Unlock}\ \ \mathbf{of}\ \ \mathsf{proc}$

```
(∀s a v p.
 clause_name "read-mem" ∧
 not_blocked  s p ∧
 (s.M  a = SOME  v) ∧
 no_pending (s.B  p)a
  ⟹
 machine_trans  s
            (EVT p (ACCESS R (LOCATION_MEM a)v))
            s) ∧

(∀s a v p.
 clause_name "read-buffer" ∧
 not_blocked  s p ∧
 (∃b₁ b₂.(s.B p = b₁ ++[(a, v)] ++b₂) ∧ no_pending  b₁ a)
  ⟹
 machine_trans  s
            (EVT p (ACCESS R (LOCATION_MEM a)v))
            s) ∧

(∀s r v p.
 clause_name "read-reg" ∧
 (*not_blocked s p /\*)
 (s.R p r = SOME  v)
  ⟹
 machine_trans  s
            (EVT p (ACCESS R (LOCATION_REG p r)v))
            s) ∧

(∀s a v p s'.
 clause_name "write-buffer" ∧
 (*not_blocked s p /\*)
 (s' =⦇ R := s.R;
        M := s.M;
        $B :=(p \mapsto (a,v) \in s.B\ p) s.B;$
        $L := s.L\rrparenthesis)$
  $\Longrightarrow$
machine_trans  $s$
            (EVT $p$ (ACCESS W (LOCATION_MEM $a$)$v$))
            $s'$) $\wedge$

($\forall s\ a\ v\ p\ b\ s'$.
clause_name "write-mem" $\wedge$
not_blocked  $s\ p \wedge$
$(s.B\ p = b ++ [(a,v)]) \wedge$
$(s' = \llparenthesis\ R := s.R;$
        $M :=(a \mapsto$ SOME  $v) s.M;$
        $B :=(p \mapsto b) s.B;$
        $L := s.L\rrparenthesis)$
  $\Longrightarrow$
machine_trans  $s$ TAU $s'$) $\wedge$

($\forall s\ r\ v\ p\ s'$.
clause_name "write-reg" $\wedge$
(*not_blocked s p*)
$(s' = \llparenthesis\ R :=(p \mapsto (r \mapsto$ SOME  $v)(s.R\ p)) s.R;$
        $M := s.M;$
        $B := s.B;$
        $L := s.L\rrparenthesis)$
  $\Longrightarrow$
machine_trans  $s$
            (EVT $p$ (ACCESS W (LOCATION_REG $p\ r$)$v$))
            $s'$) $\wedge$

($\forall s\ p$.
clause_name "barrier" $\wedge$
(*not_blocked s p /\*)
$(s.B\ p = [\,])$
  $\Longrightarrow$
machine_trans  $s$ (EVT $p$ (BARRIER MFENCE))$s$) $\wedge$

($\forall s\ p\ b$.
clause_name "nop" $\wedge$
(*not_blocked s p /\*)
$b \neq$ MFENCE
  $\Longrightarrow$
machine_trans  $s$ (EVT $p$ (BARRIER $b$))$s$) $\wedge$

($\forall s\ p\ s'$.
clause_name "lock" $\wedge$
$(s.L =$ NONE$) \wedge$
$(s.B\ p = [\,]) \wedge$
$(s' = \llparenthesis\ R := s.R;$
```

$M := s.M$;
$B := s.B$;
$L := \textsc{Some}\ \ p \rrparenthesis)$
$\Longrightarrow$
machine_trans s ($\textsc{Lock}\ p)s'$) $\wedge$

($\forall s\ p\ s'$.
clause_name "unlock" $\wedge$
($s.L = \textsc{Some}\ \ p$) $\wedge$
($s.B\ p = [\,]$) $\wedge$
($s' = \llparenthesis\ R := s.R$;
$M := s.M$;
$B := s.B$;
$L := \textsc{None}\rrparenthesis$)
$\Longrightarrow$
machine_trans s ($\textsc{Unlock}\ p)s'$)

machine_state_to_state_constraint s =
λl.
case l **of**
$\textsc{Location\_mem}\ a \rightarrow s.M\ \ a$
$\|\ \textsc{Location\_reg}\ p\ r \rightarrow s.R\ p\ r$

machine_final_state *path* =
if finite *path* **then**
$\textsc{Some}$ (machine_state_to_state_constraint (last *path*))
else
$\textsc{None}$

machine_init_state *sc* =
$\llparenthesis\ R$:=($\lambda p\ r.sc$ ($\textsc{Location\_reg}\ p\ r$));
M :=($\lambda a.sc$ ($\textsc{Location\_mem}\ a$));
B :=($\lambda p.[\,]$);
$L := \textsc{None}\rrparenthesis$

is_init $s = \exists sc.s$ = machine_init_state *sc*

evt_machine_state $= \llparenthesis$
(* Per processor registers, annotated with the event that last wrote it *)
eR : proc $\rightarrow$ $'reg$ $\rightarrow$ (value#$'reg$ event option) option;
(* main memory, annotated with the event that last wrote it *)
eM : address $\rightarrow$ (value#$'reg$ event option) option;
(* Per processor FIFO write buffers *)
eB : proc $\rightarrow$ $'reg$ event list;
(* Which processor holds the lock *)
eL : proc option

$\rrbracket$

evt_not_blocked $s\ p = (s.eL = \textsc{None}) \lor (s.eL = \textsc{Some}\ \ p)$

evt_no_pending $b\ a = \neg(\exists e.\,\text{MEM}\ \ e\ b \land (\text{loc}\ \ e = \textsc{Some}\ \ (\textsc{Location\_mem}\ a)))$

evt_machine_label =
$\textsc{TauEvt}$ **of** $'reg$ event
 | $\text{RE}\textsc{vt}$ **of** $'reg$ event $'reg$ event option
 | $\text{WE}\textsc{vt}$ **of** $'reg$ event
 | $\text{BE}\textsc{vt}$ **of** $'reg$ event
 | $\textsc{Lock}\text{E}$ **of** proc $'reg$ event set
 | $\textsc{Unlock}\text{E}$ **of** proc $'reg$ event set

$(\forall s\ a\ v\ p\ er\ ew\_opt.$
clause_name "evt-read-mem" $\land$
evt_not_blocked $s\ p \land$
$(\text{proc}\ er = p) \land$
$(er.action = \textsc{Access}\ R\ (\textsc{Location\_mem}\ a)v) \land$
$(s.eM\ \ a = \textsc{Some}\ \ (v, ew\_opt)) \land$
evt_no_pending $(s.eB\ p)a$
$\Longrightarrow$
evt_machine_trans $s\ (\text{RE}\textsc{vt}\ er\ ew\_opt)s) \land$

$(\forall s\ a\ v\ p\ er\ ew.$
clause_name "evt-read-buffer" $\land$
evt_not_blocked $s\ p \land$
$(\text{proc}\ er = p) \land$
$(er.action = \textsc{Access}\ R\ (\textsc{Location\_mem}\ a)v) \land$
$(ew.action = \textsc{Access}\ \text{W}\ (\textsc{Location\_mem}\ a)v) \land$
$(\exists b_1\ b_2.(s.eB\ p = b_1 +\!\!+ [ew] +\!\!+ b_2) \land \text{evt\_no\_pending}\ \ b_1\ a)$
$\Longrightarrow$
evt_machine_trans $s\ (\text{RE}\textsc{vt}\ er\ (\textsc{Some}\ \ ew))s) \land$

$(\forall s\ r\ v\ p\ er\ ew\_opt.$
clause_name "evt-read-reg" $\land$
(*evt_not_blocked s p /*)
$(\text{proc}\ er = p) \land$
$(er.action = \textsc{Access}\ R\ (\textsc{Location\_reg}\ p\ r)v) \land$
$(s.eR\ p\ r = \textsc{Some}\ \ (v, ew\_opt))$
$\Longrightarrow$
evt_machine_trans $s\ (\text{RE}\textsc{vt}\ er\ ew\_opt)s) \land$

$(\forall s\ a\ v\ p\ ew\ s'.$
clause_name "evt-write-buffer" $\land$
(*evt_not_blocked s p /*)
$(\text{proc}\ ew = p) \land$

$(ew.action = \textsc{Access}\ \mathrm{W}\ (\textsc{Location\_mem}\ a) v) \wedge$
$(s' = \langle\!| \ eR := s.eR;$
$\quad eM := s.eM;$
$\quad eB := (p \mapsto ew \in s.eB\ p) s.eB;$
$\quad eL := s.eL |\!\rangle)$
$\Longrightarrow$
evt_machine_trans s (WEvt ew)s') $\wedge$

$(\forall s\ a\ v\ p\ ew\ b\ s'.$
clause_name "evt-write-mem" $\wedge$
evt_not_blocked $s\ p \wedge$
$(\mathsf{proc}\ ew = p) \wedge$
$(ew.action = \textsc{Access}\ \mathrm{W}\ (\textsc{Location\_mem}\ a) v) \wedge$
$(s.eB\ p = b +\!\!+ [ew]) \wedge$
$(s' = \langle\!| \ eR := s.eR;$
$\quad eM := (a \mapsto \textsc{Some}\ \ (v, \textsc{Some}\ \ ew)) s.eM;$
$\quad eB := (p \mapsto b) s.eB;$
$\quad eL := s.eL |\!\rangle)$
$\Longrightarrow$
evt_machine_trans s (TauEvt ew)s') $\wedge$

$(\forall s\ r\ v\ p\ ew\ s'.$
clause_name "evt-write-reg" $\wedge$
(*evt_not_blocked s p /*)
$(\mathsf{proc}\ ew = p) \wedge$
$(ew.action = \textsc{Access}\ \mathrm{W}\ (\textsc{Location\_reg}\ p\ r) v) \wedge$
$(s' = \langle\!| \ eR := (p \mapsto (r \mapsto \textsc{Some}\ \ (v, \textsc{Some}\ \ ew))(s.eR\ p)) s.eR;$
$\quad eM := s.eM;$
$\quad eB := s.eB;$
$\quad eL := s.eL |\!\rangle)$
$\Longrightarrow$
evt_machine_trans s (WEvt ew)s') $\wedge$

$(\forall s\ p\ eb.$
clause_name "evt-barrier" $\wedge$
(*evt_not_blocked s p /*)
$(\mathsf{proc}\ eb = p) \wedge$
$(eb.action = \textsc{Barrier}\ \textsc{Mfence}) \wedge$
$(s.eB\ p = [\,])$
$\Longrightarrow$
evt_machine_trans s (BEvt eb)s) $\wedge$

$(\forall s\ eb.$
clause_name "evt-nop" $\wedge$
(*not_blocked s p /*)
$((eb.action = \textsc{Barrier}\ \textsc{Sfence}) \vee (eb.action = \textsc{Barrier}\ \textsc{Lfence}))$
$\Longrightarrow$
evt_machine_trans s (BEvt eb)s) $\wedge$

$(\forall s\ p\ s'\ es.$
clause_name "evt-lock" $\wedge$
$(s.eL = \textsc{None}) \wedge$
$(s.eB\ p = [\,]) \wedge$
$(s' = \langle\!|\ eR := s.eR;$
$\quad eM := s.eM;$
$\quad eB := s.eB;$
$\quad eL := \textsc{Some}\ \ p|\!\rangle)$
$\Longrightarrow$
evt_machine_trans s (LOCKE $p\ es)s')\ \wedge$

$(\forall s\ p\ s'\ es.$
clause_name "evt-unlock" $\wedge$
$(s.eL = \textsc{Some}\ \ p) \wedge$
$(s.eB\ p = [\,]) \wedge$
$(s' = \langle\!|\ eR := s.eR;$
$\quad eM := s.eM;$
$\quad eB := s.eB;$
$\quad eL := \textsc{None}|\!\rangle)$
$\Longrightarrow$
evt_machine_trans s (UNLOCKE $p\ es)s')$

evt_machine_state_to_state_constraint s =
$\lambda l.$
case l **of**
$\quad$ LOCATION_MEM $a \rightarrow$ OPTION_MAP FST $(s.eM\ a)$
$\|$ LOCATION_REG $p\ r \rightarrow$ OPTION_MAP FST $(s.eR\ p\ r)$

evt_machine_final_state *path* =
if finite *path* **then**
SOME (evt_machine_state_to_state_constraint (last *path*))
else
NONE

evt_machine_init_state sc =
$\langle\!|\ eR := (\lambda p\ r.$ OPTION_MAP $(\lambda v.(v, \textsc{None}))(sc$ (LOCATION_REG $p\ r)));$
$\quad eM := (\lambda a.$ OPTION_MAP $(\lambda v.(v, \textsc{None}))(sc$ (LOCATION_MEM $a)));$
$\quad eB := (\lambda p.[\,]);$
$\quad eL := \textsc{None}|\!\rangle$

evt_is_init $s = \exists sc.s =$ evt_machine_init_state sc

(get_orig_event (REVT e _) = SOME $e) \wedge$
(get_orig_event (WEVT e) = SOME $e) \wedge$
(get_orig_event (BEVT e) = SOME $e) \wedge$
(get_orig_event _ = NONE)

locked_segment $path\ i\ j\ p =$
$j + 1 \ \in\ \mathrm{PL}\ path \wedge$
$i < j \wedge$
$(\exists es.\ \mathrm{nth\_label}\ i\ path = \textsc{Lock}\mathrm{E}\ p\ es) \wedge$
$(\exists es.\ \mathrm{nth\_label}\ j\ path = \textsc{Unlock}\mathrm{E}\ p\ es) \wedge$
$(\forall k\ es. i < k \wedge k < j \implies \mathrm{nth\_label}\ k\ path \neq \textsc{Unlock}\mathrm{E}\ p\ es)$

okEpath $E\ path =$
(* The REvt, WEvt and BEvt labels are exactly the set of events *)
$(E.events = \{e \mid \exists i. i + 1 \ \in\ \mathrm{PL}\ path \wedge (\mathrm{get\_orig\_event}\ (\mathrm{nth\_label}\ i\ path) = \textsc{Some}\ \ e)\}) \wedge$
(* No REvt, WEvt, or BEvt appears twice as a label *)
$(\forall i\ j\ e_1\ e_2.$
$\quad i + 1 \ \in\ \mathrm{PL}\ path \wedge j + 1 \ \in\ \mathrm{PL}\ path \wedge$
$\quad (\mathrm{get\_orig\_event}\ (\mathrm{nth\_label}\ i\ path) = \textsc{Some}\ \ e_1) \wedge (\mathrm{get\_orig\_event}\ (\mathrm{nth\_label}\ j\ path) = \textsc{Some}\ \ e_2) \wedge$
$\quad (e_1 = e_2)$
$\quad\ \implies$
$\quad (i = j)) \wedge$
(* The REvt, WEvt, and BEvt parts of the trace follow po_iico *)
$(\forall (e_1, e_2) \in (\mathrm{po\_iico}\ \ E). \exists i\ j.$
$\quad i < j \wedge j + 1 \ \in\ \mathrm{PL}\ path \wedge$
$\quad (\mathrm{get\_orig\_event}\ (\mathrm{nth\_label}\ i\ path) = \textsc{Some}\ \ e_1) \wedge (\mathrm{get\_orig\_event}\ (\mathrm{nth\_label}\ j\ path) = \textsc{Some}\ \ e_2)) \wedge$
(* atomic sets of events are properly bracketed by lock/unlock pairs *)
$(\forall es \in (E.atomicity).$
$\quad \exists i\ j\ p.$
$\qquad \mathrm{locked\_segment}\ path\ i\ j\ p \wedge$
$\qquad (\{e \mid e \ \in\ es \wedge e \ \in\ \mathrm{mem\_accesses}\ \ E\}$
$\qquad =$
$\qquad \{e \mid \exists k. i < k \wedge k < j \wedge$
$\qquad\qquad (\mathrm{get\_orig\_event}\ (\mathrm{nth\_label}\ k\ path) = \textsc{Some}\ \ e) \wedge$
$\qquad\qquad e \ \in\ \mathrm{mem\_accesses}\ E \wedge$
$\qquad\qquad (\mathsf{proc}\ e = p)\}))$

okMpath $path =$
evt_is_init (first $path$) $\wedge$
okpath evt_machine_trans $path \wedge$
$\forall i\ e.$
$i + 1 \ \in\ \mathrm{PL}\ path \wedge (\mathrm{nth\_label}\ i\ path = \mathrm{W}\textsc{Evt}\ e) \wedge \mathrm{is\_mem\_access}\ \ e$
$\ \implies$
$\exists j. j + 1 \ \in\ \mathrm{PL}\ path \wedge i < j \wedge (\mathrm{nth\_label}\ j\ path = \textsc{TauEvt}\ e)$

erase_state $s\ s' =$
$(\forall p\ r. s'.R\ p\ r = \mathrm{OPTION\_MAP}\ \ \mathrm{FST}\ (s.eR\ p\ r)) \wedge$
$(\forall a. s'.M\ \ a = \mathrm{OPTION\_MAP}\ \ \mathrm{FST}\ (s.eM\ \ a)) \wedge$
$(\forall p. (\mathrm{LENGTH}\ (s'.B\ p) = \mathrm{LENGTH}\ (s.eB\ p)) \wedge$
$\quad \forall n. n < \mathrm{LENGTH}\ (s.eB\ p) \implies$
$\qquad \exists e\ a\ v. (\mathrm{EL}\ \ n\ (s.eB\ p) = e) \wedge$
$\qquad\qquad (\mathsf{proc}\ e = p) \wedge$
$\qquad\qquad (e.action = \textsc{Access}\ \mathrm{W}\ (\textsc{Location\_mem}\ a) v) \wedge$

$(\mathrm{EL}\ n\ (s'.B\ p) = (a, v))) \wedge$
$(s'.L = s.eL)$

(erase_label ($\textsc{TauEvt}$ _) = $\textsc{Tau}$) $\wedge$
(erase_label (R$\textsc{Evt}$ e _) = $\textsc{Evt}$ (proc e)$e.action$) $\wedge$
(erase_label (W$\textsc{Evt}$ e) = $\textsc{Evt}$ (proc e)$e.action$) $\wedge$
(erase_label (B$\textsc{Evt}$ e) = $\textsc{Evt}$ (proc e)$e.action$) $\wedge$
(erase_label ($\textsc{Lock}$E p es) = $\textsc{Lock}$ p) $\wedge$
(erase_label ($\textsc{Unlock}$E p es) = $\textsc{Unlock}$ p)

(annotated_labels $\textsc{Tau}$ ew e_opt = $\{\textsc{TauEvt}\ ew\}$) $\wedge$
(annotated_labels ($\textsc{Evt}$ p ($\textsc{Access}$ R l v))ew e_opt =
  $\{$R$\textsc{Evt}$ e e_opt | e | ($e.action$ = $\textsc{Access}$ R l v) $\wedge$ (p = proc e)$\}$) $\wedge$
(annotated_labels ($\textsc{Evt}$ p ($\textsc{Access}$ W l v))ew e_opt =
  $\{$W$\textsc{Evt}$ e | ($e.action$ = $\textsc{Access}$ W l v) $\wedge$ (p = proc e)$\}$) $\wedge$
(annotated_labels ($\textsc{Evt}$ p ($\textsc{Barrier}$ b))ew e_opt =
  $\{$B$\textsc{Evt}$ e | ($e.action$ = $\textsc{Barrier}$ b) $\wedge$ (p = proc e)$\}$) $\wedge$
(annotated_labels ($\textsc{Lock}$ p)ew e_opt =
  $\{\textsc{Lock}$E p es | es | $\mathbf{T}\}$) $\wedge$
(annotated_labels ($\textsc{Unlock}$ p)ew e_opt =
  $\{\textsc{Unlock}$E p es | es | $\mathbf{T}\}$)

Part V

lts_erasure

Theorem 1

$\forall sc.\,$erase_state (evt_machine_init_state sc)(machine_init_state sc)

Theorem 2

$\forall s\ s'.$
erase_state $s\ s'$
$\implies$
(evt_machine_state_to_state_constraint s =
machine_state_to_state_constraint s')

Theorem 3

$\forall l\ l'\ ew\ er\_opt.$
$l\ \in$ annotated_labels $l'\ ew\ er\_opt$
$\implies$
(l' = erase_label l)

Theorem 4

$\forall s_1\ l\ s_2\ s_1'.$
evt_machine_trans $s_1\ l\ s_2\ \wedge$
erase_state $s_1\ s_1'$
$\implies$
$\exists s_2'.$
erase_state $s_2\ s_2'\ \wedge$
machine_trans s_1' (erase_label l)s_2'

Theorem 5

$\forall s_1'\ l'\ s_2'\ s_1.$
machine_trans $s_1'\ l'\ s_2'\ \wedge$
erase_state $s_1\ s_1'$
$\implies$
$\exists ew\ ew\_opt.$
$\forall l.l\ \in$ annotated_labels $l'\ ew\ ew\_opt\ \implies$
$\exists s_2.\,$erase_state $s_2\ s_2'\ \wedge$ evt_machine_trans $s_1\ l\ s_2$

Part VI

linear_valid_execution

Theorem 6

$\forall E\ X.\,$ linear_valid_execution $E\ X \implies$ valid_execution $E\ X$

Theorem 7

$\forall E\ X\ \mathit{memory\_order}'.$
valid_execution $E\ X \wedge$
$X.\mathit{memory\_order} \subseteq \mathit{memory\_order}' \wedge$
linear_order $\mathit{memory\_order}'$ (mem_accesses E) $\wedge$
finite_prefixes $\mathit{memory\_order}'$ (mem_accesses E) $\wedge$
$(\forall er \in$ (mem_reads E)$.\forall ew \in$ (mem_writes E).
(loc $er =$ loc ew) $\wedge (ew, er) \in \mathit{memory\_order}' \implies (ew, er) \in X.\mathit{memory\_order})$
$\implies$
linear_valid_execution $E\ (X \oplus \mathit{memory\_order} := \mathit{memory\_order}')$

complete_memory_order $E\ \mathit{memory\_order} =$
$\mathit{memory\_order} \cup$
$\{(e_1, e_2) \mid \exists ew\ er.(ew, er) \notin \mathit{memory\_order} \wedge (er, ew) \notin \mathit{memory\_order} \wedge$
$ew \in$ mem_writes $E \wedge er \in$ mem_reads $E \wedge$ (loc $ew =$ loc er) $\wedge$
$(e_1, er) \in \mathit{memory\_order} \wedge (ew, e_2) \in \mathit{memory\_order}\}$

Theorem 8

$\forall E\ X.$
well_formed_event_structure $E \wedge$
valid_execution $E\ X$
$\implies$
$\exists \mathit{memory\_order}'.$
$X.\mathit{memory\_order} \subseteq \mathit{memory\_order}' \wedge$
linear_order $\mathit{memory\_order}'$ (mem_accesses E) $\wedge$
finite_prefixes $\mathit{memory\_order}'$ (mem_accesses E) $\wedge$
$(\forall er \in$ (mem_reads E)$.\forall ew \in$ (mem_writes E).
(loc $er =$ loc ew) $\wedge (ew, er) \in \mathit{memory\_order}' \implies (ew, er) \in X.\mathit{memory\_order})$

Part VII

lts_trace

no_dup_writes $path =$
$\forall i\ j\ ew.$
SUC $i \in$ PL $path \land$
SUC $j \in$ PL $path \land$
(nth_label $i\ path =$ WEvt $ew) \land$
(nth_label $j\ path =$ WEvt $ew)$
$\Longrightarrow$
$(i = j)$

Theorem 9

$\forall path\ j\ e.$
okMpath $path \land$
$j + 1 \in$ PL $path \land$
(nth_label $j\ path =$ TauEvt $e)$
$\Longrightarrow$
$\exists i.i < j \land$ (nth_label $i\ path =$ WEvt $e) \land$ is_mem_access e

Theorem 10

$\forall path\ i\ j\ ew_1\ ew_2.$
okMpath $path \land$
$j + 1 \in$ PL $path \land$
$i < j \land$
(proc $ew_1 =$ proc $ew_2) \land$
(nth_label $i\ path =$ TauEvt $ew_1) \land$
(nth_label $j\ path =$ TauEvt $ew_2)$
$\Longrightarrow$
$\exists k\ l.k < i \land l < j \land k < l \land$
(nth_label $k\ path =$ WEvt $ew_1) \land$ (nth_label $l\ path =$ WEvt $ew_2)$

Theorem 11

$\forall path\ i\ e_1\ e_2.$
okMpath $path \land$
$i + 1 \in$ PL $path \land$
(nth_label $i\ path =$ REvt $e_1\ e_2)$
$\Longrightarrow$
$\exists l\ v.e_1.action =$ Access $R\ l\ v$

Theorem 12

$\forall path\ i\ e.$
okMpath $path \land$
$i + 1 \in$ PL $path \land$
(nth_label $i\ path =$ WEvt $e)$
$\Longrightarrow$
$\exists l\ v.e.action =$ Access W $l\ v$

Theorem 13

$\forall path\ i\ e.$
okMpath $path \land$

$i+1 \in \mathrm{PL}\ \mathit{path} \wedge$
$(\mathrm{nth\_label}\ i\ \mathit{path} = \mathrm{BE}\textsc{vt}\ e)$
$\Longrightarrow$
$\exists f.e.\mathit{action} = \textsc{Barrier}\, f$

Theorem 14

$\forall \mathit{path}\ i\ e.$
$\mathrm{okMpath}\ \mathit{path} \wedge$
$i+1 \in \mathrm{PL}\ \mathit{path} \wedge$
$(\mathrm{nth\_label}\ i\ \mathit{path} = \mathrm{WE}\textsc{vt}\ e) \wedge$
$\mathrm{is\_mem\_access}\ e$
$\Longrightarrow$
$\exists j.j+1 \in \mathrm{PL}\ \mathit{path} \wedge i < j \wedge (\mathrm{nth\_label}\ j\ \mathit{path} = \textsc{TauEvt}\ e)$

Theorem 15

$\forall \mathit{path}\ i\ j\ ew_1\ ew_2.$
$\mathrm{okMpath}\ \mathit{path} \wedge$
$j+1 \in \mathrm{PL}\ \mathit{path} \wedge$
$i < j \wedge$
$(\mathsf{proc}\ ew_1 = \mathsf{proc}\ ew_2) \wedge$
$(\mathrm{nth\_label}\ i\ \mathit{path} = \mathrm{WE}\textsc{vt}\ ew_1) \wedge$
$(\mathrm{nth\_label}\ j\ \mathit{path} = \mathrm{WE}\textsc{vt}\ ew_2) \wedge$
$\mathrm{is\_mem\_access}\ ew_1 \wedge$
$\mathrm{is\_mem\_access}\ ew_2 \wedge$
$\mathrm{no\_dup\_writes}\ \mathit{path}$
$\Longrightarrow$
$\exists k\ l.l+1 \in \mathrm{PL}\ \mathit{path} \wedge k < l \wedge (\mathrm{nth\_label}\ k\ \mathit{path} = \textsc{TauEvt}\ ew_1) \wedge (\mathrm{nth\_label}\ l\ \mathit{path} = \textsc{TauEvt}\ ew_2)$

Theorem 16

$\forall \mathit{path}\ i\ j\ ef\ ew\ p\ es.$
$\mathrm{okMpath}\ \mathit{path} \wedge$
$j+1 \in \mathrm{PL}\ \mathit{path} \wedge$
$i < j \wedge$
$(((\mathrm{nth\_label}\ j\ \mathit{path} = \mathrm{BE}\textsc{vt}\ ef) \wedge (\mathsf{proc}\ ef = \mathsf{proc}\ ew) \wedge (ef.\mathit{action} = \textsc{Barrier Mfence})) \vee$
$(\mathrm{nth\_label}\ j\ \mathit{path} = \textsc{UnlockE}\ (\mathsf{proc}\ ew) es) \vee$
$(\mathrm{nth\_label}\ j\ \mathit{path} = \textsc{LockE}\ (\mathsf{proc}\ ew) es)) \wedge$
$(\mathrm{nth\_label}\ i\ \mathit{path} = \mathrm{WE}\textsc{vt}\ ew) \wedge$
$\mathrm{is\_mem\_access}\ ew$
$\Longrightarrow$
$\exists k.k < j \wedge (\mathrm{nth\_label}\ k\ \mathit{path} = \textsc{TauEvt}\ ew)$

Theorem 17

$\forall \mathit{path}\ j\ er\ ew.$
$\mathrm{okMpath}\ \mathit{path} \wedge$
$j+1 \in \mathrm{PL}\ \mathit{path} \wedge$
$\mathrm{no\_dup\_writes}\ \mathit{path} \wedge$
$(\mathrm{nth\_label}\ j\ \mathit{path} = \mathrm{RE}\textsc{vt}\ er\ (\textsc{Some}\ \ ew))$
$\Longrightarrow$

$(\text{loc}\ er = \text{loc}\ ew) \wedge$
$(\text{value\_of}\ er = \text{value\_of}\ ew) \wedge$
$(((\mathsf{proc}\ er = \mathsf{proc}\ ew) \wedge$
$\exists i.$
$i < j \wedge$
$(\text{nth\_label}\ i\ path = \text{WE}\textsc{vt}\ ew) \wedge$
$(\forall k. i < k \wedge k < j \implies \text{nth\_label}\ k\ path \neq \text{T}\textsc{au}\text{E}\textsc{vt}\ ew) \wedge$
$(\forall k\ ew'. i < k \wedge k < j \wedge (\text{nth\_label}\ k\ path = \text{WE}\textsc{vt}\ ew') \wedge (\mathsf{proc}\ er = \mathsf{proc}\ ew') \implies$
$\quad\quad \text{loc}\ ew \neq \text{loc}\ ew')) \vee$
$(\exists i.$
$i < j \wedge$
$(\text{nth\_label}\ i\ path = \text{T}\textsc{au}\text{E}\textsc{vt}\ ew) \wedge$
$(\forall k\ ew'. k < j \wedge (\text{nth\_label}\ k\ path = \text{WE}\textsc{vt}\ ew') \wedge (\mathsf{proc}\ er = \mathsf{proc}\ ew') \wedge$
$\quad\quad (\text{loc}\ ew' = \text{loc}\ er) \implies$
$\quad\quad \exists l. k < l \wedge l < j \wedge (\text{nth\_label}\ l\ path = \text{T}\textsc{au}\text{E}\textsc{vt}\ ew')) \wedge$
$(\forall k\ ew'. i < k \wedge k < j \wedge (\text{nth\_label}\ k\ path = \text{T}\textsc{au}\text{E}\textsc{vt}\ ew') \implies \text{loc}\ ew \neq \text{loc}\ ew')))$

Theorem 18
$\forall path\ i\ er.$
$\text{okMpath}\ path \wedge$
$i + 1 \in \text{PL}\ path \wedge$
$(\text{nth\_label}\ i\ path = \text{RE}\textsc{vt}\ er\ \text{N}\textsc{one})$
$\implies$
$(\text{value\_of}\ er = \text{evt\_machine\_state\_to\_state\_constraint}\ (\text{first}\ path)(\textbf{the}\ (\text{loc}\ er))) \wedge$
$\forall j\ ew.$
$j < i \wedge$
$((\text{nth\_label}\ j\ path = \text{T}\textsc{au}\text{E}\textsc{vt}\ ew) \vee ((\text{nth\_label}\ j\ path = \text{WE}\textsc{vt}\ ew) \wedge (\mathsf{proc}\ er = \mathsf{proc}\ ew)))$
$\implies$
$\text{loc}\ ew \neq \text{loc}\ er$

Theorem 19
$\forall path\ i\ j\ k\ p\ e\ e'.$
$\text{okMpath}\ path \wedge$
$\text{locked\_segment}\ path\ i\ k\ p \wedge$
$i < j \wedge$
$j < k \wedge$
$\text{is\_mem\_access}\ e \wedge$
$((\text{nth\_label}\ j\ path = \text{RE}\textsc{vt}\ e\ e') \vee$
$(\text{nth\_label}\ j\ path = \text{T}\textsc{au}\text{E}\textsc{vt}\ e))$
$\implies$
$(\mathsf{proc}\ e = p)$

Theorem 20
$\forall path\ i\ j\ ew.$
$\text{okMpath}\ path \wedge$
$\text{locked\_segment}\ path\ i\ j\ (\mathsf{proc}\ ew) \wedge$
$\text{is\_mem\_access}\ ew$
$\implies$
$((\exists k. i < k \wedge k < j \wedge (\text{nth\_label}\ k\ path = \text{WE}\textsc{vt}\ ew)) =$
$(\exists k. i < k \wedge k < j \wedge (\text{nth\_label}\ k\ path = \text{T}\textsc{au}\text{E}\textsc{vt}\ ew)))$

Part VIII

lts_axiomatic_equiv

```
(get_mem_event (TauEvt $e$) = Some  $e$) $\wedge$
(get_mem_event (REvt $e$ _) =
   if  is_mem_access  $e$  then
      Some  $e$
    else
      None) $\wedge$
(get_mem_event  _ = None)
```

```
path_to_X  $path$ =
$\langle\!|$ $memory\_order$ :=
  $\{(e_1, e_2) \mid$
  $\exists j\ i\ l_1\ l_2.$
     $j + 1 \in$ PL $path \wedge i \leq j \wedge$
     (nth_label $i$ $path$ = $l_1$) $\wedge$ (nth_label $j$ $path$ = $l_2$) $\wedge$
     (get_mem_event $l_1$ = Some $e_1$) $\wedge$ (get_mem_event $l_2$ = Some $e_2$)$\}$;
  $rfmap$ :=$\{(ew, er) \mid (ew, er) \mid \exists i. i + 1 \in$ PL $path \wedge$ (nth_label $i$ $path$ = REvt $er$ (Some $ew$))$\}$;
  $initial\_state$ := evt_machine_state_to_state_constraint (first $path$)$|\!\rangle$
```

Theorem 21

```
$\forall E\ path.$
 well_formed_event_structure  $E$ $\wedge$
 okEpath  $E$ $path$ $\wedge$
 okMpath  $path$
  $\Longrightarrow$
 linear_valid_execution  $E$ (path_to_X  $path$)
```

```
memL  $E$ $X$ =
$\{$TauEvt $e \mid e \in$ mem_writes $E\} \cup$
$\{$REvt $er$ None $\mid er \in$ mem_reads $E \wedge er \notin$ range $X.rfmap\} \cup$
$\{$REvt $er$ (Some $ew$) $\mid er \in$ mem_reads $E \wedge (ew, er) \in X.rfmap\}$
```

```
to_memL  $E$ $X$ $e$ =
if  $e \in$ mem_writes $E$  then
TauEvt $e$
 else  if  $e \in$ mem_reads $E \wedge e \notin$ range $X.rfmap$  then
REvt $e$ None
 else
REvt $e$ (Some  ($CHOICE\{ew \mid (ew, e) \in X.rfmap\}$))
```

```
localL  $E$ $X$ =
$\{$REvt $er$ None $\mid er \in$ reads $E \wedge er \notin$ range $X.rfmap\} \cup$
$\{$REvt $er$ (Some $ew$) $\mid er \in$ reads $E \wedge (ew, er) \in X.rfmap\} \cup$
$\{$WEvt $e \mid e \in$ writes $E\} \cup$
$\{$BEvt $e \mid e \in$ fences $E\}$
```

```
to_localL  $E$ $X$ $e$ =
```

if $e \in$ writes E **then**
WEvt e
else if $e \in$ fences E **then**
BEvt e
else if $e \in$ reads $E \wedge e \notin$ range $X.\mathit{rfmap}$ **then**
REvt e None
else
REvt e (Some $(\mathit{CHOICE}\{ew \mid (ew, e) \in X.\mathit{rfmap}\})$)

proc_es $es = \{\mathsf{proc}\ e \mid e \in es\}$

lockL $E\ X =$
$\{$LockE $p\ es \mid es \in E.\mathit{atomicity} \wedge (p \in$ proc_es $es)\} \cup$
$\{$UnlockE $p\ es \mid es \in E.\mathit{atomicity} \wedge (p \in$ proc_es $es)\}$

allL $E\ X =$
memL $E\ X\ \cup$
localL $E\ X\ \cup$
lockL $E\ X$

(L_e (TauEvt e) = Some e) $\wedge$
(L_e (REvt e _) = Some e) $\wedge$
(L_e (WEvt e) = Some e) $\wedge$
(L_e (BEvt e) = Some e) $\wedge$
(L_e _ = None)

(L_es (LockE $p\ es$) = Some es) $\wedge$
(L_es (UnlockE $p\ es$) = Some es) $\wedge$
(L_es _ = None)

lo1 $E\ X =$
$\{(l, l') \mid l \in$ memL $E\ X \wedge l' \in$ memL $E\ X \wedge$
  (**the** (L_e l), **the** (L_e l')) $\in X.\mathit{memory\_order}\}$

lo1_alt $E\ X =$
$\{$(to_memL $E\ X\ e$, to_memL $E\ X\ e'$) $\mid (e, e') \mid (e, e') \in X.\mathit{memory\_order}\}$

lo2 $E\ X =$
$\{(l, l') \mid l \in$ localL $E\ X \wedge l' \in$ localL $E\ X \wedge$
  (**the** (L_e l), **the** (L_e l')) $\in$ po_iico $E\} \cup$
$\{$(WEvt e, WEvt e') $\mid$ WEvt $e \in$ allL $E\ X \wedge$ WEvt $e' \in$ allL $E\ X \wedge$
    $(e, e') \in X.\mathit{memory\_order} \wedge (\mathsf{proc}\ e = \mathsf{proc}\ e')\} \cup$
$\{(l,$ WEvt $e) \mid l \in$ localL $E\ X \wedge$ WEvt $e \in$ allL $E\ X \wedge$
  $\exists e'.$WEvt $e' \in$ localL $E\ X \wedge$ (**the** (L_e l), e') $\in$ po_iico $E \wedge$
    $(e', e) \in X.\mathit{memory\_order} \wedge (\mathsf{proc}\ e' = \mathsf{proc}\ e)\}$

$\text{lo2\_alt}\ E\ X =$
$\{(\text{to\_localL}\ E\ X\ e, \text{to\_localL}\ E\ X\ e') \mid (e,e') \mid (e,e') \in \text{po\_iico}\ E\} \cup$
$\{(\text{WEvt}\ e, \text{WEvt}\ e') \mid \text{WEvt}\ e \in \text{allL}\ E\ X \wedge \text{WEvt}\ e' \in \text{allL}\ E\ X \wedge$
$\quad (e,e') \in X.\mathit{memory\_order} \wedge (\mathsf{proc}\ e = \mathsf{proc}\ e')\} \cup$
$\{(\text{to\_localL}\ E\ X\ e'', \text{WEvt}\ e) \mid (e'',e) \mid$
$\quad \text{WEvt}\ e \in \text{allL}\ E\ X \wedge$
$\quad \exists e'.\text{WEvt}\ e' \in \text{localL}\ E\ X \wedge (e'',e') \in \text{po\_iico}\ E \wedge$
$\qquad (e',e) \in X.\mathit{memory\_order} \wedge (\mathsf{proc}\ e' = \mathsf{proc}\ e)\}$

$\text{lo3}\ E\ X =$
$\{(\text{WEvt}\ e, \text{TauEvt}\ e) \mid \text{WEvt}\ e \in \text{allL}\ E\ X \wedge \text{TauEvt}\ e \in \text{allL}\ E\ X\}$

$\text{lo4}\ E\ X =$
$\{(\text{TauEvt}\ e, \text{BEvt}\ e') \mid \text{TauEvt}\ e \in \text{allL}\ E\ X \wedge \text{BEvt}\ e' \in \text{allL}\ E\ X \wedge$
$\quad e' \in \text{mfences}\ E \wedge (e,e') \in \text{po\_iico}\ E\}$

$\text{lo5}\ E\ X =$
$\{(\text{LockE}\ p\ es, \text{LockE}\ p\ es) \mid \text{LockE}\ p\ es \in \text{lockL}\ E\ X\} \cup$
$\{(\text{UnlockE}\ p\ es, \text{UnlockE}\ p\ es) \mid \text{LockE}\ p\ es \in \text{lockL}\ E\ X\} \cup$
$\{(l1, l2) \mid l1 \in \text{lockL}\ E\ X \wedge l2 \in \text{lockL}\ E\ X \wedge$
$\quad \textbf{the}\ (\text{L\_es}\ l1) \neq \textbf{the}\ (\text{L\_es}\ l2) \wedge$
$\quad \exists e_1\ e_2.e_1 \in \textbf{the}\ (\text{L\_es}\ l1) \wedge e_2 \in \textbf{the}\ (\text{L\_es}\ l2) \wedge$
$\qquad (e_1,e_2) \in X.\mathit{memory\_order}\} \cup$
$\{(\text{LockE}\ p\ es, \text{UnlockE}\ p\ es) \mid \text{LockE}\ p\ es \in \text{lockL}\ E\ X \wedge \text{UnlockE}\ p\ es \in \text{lockL}\ E\ X\}$

$\text{lo6}\ E\ X =$
$\{(\text{LockE}\ p\ es, l) \mid \text{LockE}\ p\ es \in \text{allL}\ E\ X \wedge l \in \text{allL}\ E\ X \wedge$
$\quad \exists e.(\text{L\_e}\ l = \text{Some}\ e) \wedge e \in es \wedge e \in \text{mem\_accesses}\ E\}$

$\text{lo7}\ E\ X =$
$\{(l, \text{UnlockE}\ p\ es) \mid l \in \text{memL}\ E\ X \wedge \text{UnlockE}\ p\ es \in \text{allL}\ E\ X \wedge$
$\quad \exists e.(\text{L\_e}\ l = \text{Some}\ e) \wedge e \in es\}$

$\text{lo8}\ E\ X =$
$\{(\text{UnlockE}\ p\ es, l) \mid \text{UnlockE}\ p\ es \in \text{allL}\ E\ X \wedge l \in \text{allL}\ E\ X \wedge$
$\quad \exists e\ e'.(\text{L\_e}\ l = \text{Some}\ e) \wedge e' \in es \wedge e \notin es \wedge$
$\qquad (e',e) \in X.\mathit{memory\_order} \wedge$
$\qquad (l \notin \text{memL}\ E\ X \implies ((\mathsf{proc}\ e = p) \wedge e \in \text{mem\_accesses}\ E))\}$

$\text{lo9}\ E\ X =$
$\{(l, \text{LockE}\ p\ es) \mid l \in \text{memL}\ E\ X \wedge \text{LockE}\ p\ es \in \text{allL}\ E\ X \wedge$
$\quad \exists e\ e'.(\text{L\_e}\ l = \text{Some}\ e) \wedge e' \in es \wedge e \notin es \wedge$
$\qquad (e,e') \in X.\mathit{memory\_order}\}$

$\text{lo\_events}\ E\ X =$

$\mathrm{lo1}\ E\ X\ \cup\ \mathrm{lo2}\ E\ X\ \cup$
$\mathrm{lo1}\ E\ X \circ \mathrm{lo2}\ E\ X \cup$
$\mathrm{lo2}\ E\ X \circ \mathrm{lo1}\ E\ X \cup$
$\mathrm{lo2}\ E\ X \circ \mathrm{lo1}\ E\ X \circ \mathrm{lo2}\ E\ X \cup$
$\mathrm{lo2}\ E\ X \circ \mathrm{lo3}\ E\ X \circ \mathrm{lo1}\ E\ X \cup$
$\mathrm{lo2}\ E\ X \circ \mathrm{lo3}\ E\ X \circ \mathrm{lo1}\ E\ X \circ \mathrm{lo2}\ E\ X \cup$
$\mathrm{lo1}\ E\ X \circ \mathrm{lo4}\ E\ X \circ \mathrm{lo2}\ E\ X \cup$
$\mathrm{lo2}\ E\ X \circ \mathrm{lo1}\ E\ X \circ \mathrm{lo4}\ E\ X \circ \mathrm{lo2}\ E\ X \cup$
$\mathrm{lo2}\ E\ X \circ \mathrm{lo3}\ E\ X \circ \mathrm{lo1}\ E\ X \circ \mathrm{lo4}\ E\ X \circ \mathrm{lo2}\ E\ X$

$\mathrm{lo68}\ E\ X = \mathrm{lo6}\ E\ X\ \cup\ \mathrm{lo8}\ E\ X$

$\mathrm{lo79}\ E\ X = \mathrm{lo7}\ E\ X\ \cup\ \mathrm{lo9}\ E\ X$

$\mathrm{lo}\ E\ X =$
$\mathrm{lo\_events}\ E\ X\ \cup$
$\mathrm{lo5}\ E\ X\ \cup$
$\mathrm{lo5}\ E\ X \circ \mathrm{lo68}\ E\ X \circ \mathrm{lo\_events}\ E\ X \cup$
$\mathrm{lo5}\ E\ X \circ \mathrm{lo68}\ E\ X \circ \mathrm{lo\_events}\ E\ X \cup$
$\mathrm{lo\_events}\ E\ X \circ \mathrm{lo79}\ E\ X \circ \mathrm{lo5}\ E\ X \cup$
$\mathrm{lo\_events}\ E\ X \circ \mathrm{lo79}\ E\ X \circ \mathrm{lo5}\ E\ X \circ \mathrm{lo68}\ E\ X \circ \mathrm{lo\_events}\ E\ X$

$\mathrm{label\_order}\ E\ X =$
$\{(l, l') \mid l\ \in\ \mathrm{memL}\ E\ X \wedge l'\ \in\ \mathrm{memL}\ E\ X \wedge$
$\quad (\mathbf{the}\ (\mathrm{L\_e}\ l), \mathbf{the}\ (\mathrm{L\_e}\ l')) \in\ X.\mathit{memory\_order}\} \cup$
$\{(l, l') \mid l\ \in\ \mathrm{localL}\ E\ X \wedge l'\ \in\ \mathrm{localL}\ E\ X \wedge$
$\quad (\mathbf{the}\ (\mathrm{L\_e}\ l), \mathbf{the}\ (\mathrm{L\_e}\ l')) \in\ \mathrm{po\_iico}\ E\} \cup$
$\{(\mathrm{WE}\textsc{vt}\ e, \mathrm{T}\textsc{au}\mathrm{E}\textsc{vt}\ e) \mid \mathrm{WE}\textsc{vt}\ e\ \in\ \mathrm{allL}\ E\ X \wedge \mathrm{T}\textsc{au}\mathrm{E}\textsc{vt}\ e\ \in\ \mathrm{allL}\ E\ X\} \cup$
$\{(\mathrm{T}\textsc{au}\mathrm{E}\textsc{vt}\ e, \mathrm{BE}\textsc{vt}\ e') \mid \mathrm{T}\textsc{au}\mathrm{E}\textsc{vt}\ e\ \in\ \mathrm{allL}\ E\ X \wedge \mathrm{BE}\textsc{vt}\ e'\ \in\ \mathrm{allL}\ E\ X \wedge$
$\quad e'\ \in\ \mathrm{mfences}\ E \wedge (e, e') \in\ \mathrm{po\_iico}\ E\} \cup$
$\{(\mathrm{T}\textsc{au}\mathrm{E}\textsc{vt}\ e, \mathrm{L}\textsc{ock}\mathrm{E}\ p\ es) \mid \mathrm{T}\textsc{au}\mathrm{E}\textsc{vt}\ e\ \in\ \mathrm{allL}\ E\ X \wedge \mathrm{L}\textsc{ock}\mathrm{E}\ p\ es\ \in\ \mathrm{allL}\ E\ X \wedge$
$\quad e\ \notin\ es \wedge \exists e'.e'\ \in\ es \wedge e'\ \in\ \mathrm{mem\_accesses}\ E \wedge (e, e') \in\ \mathrm{po\_iico}\ E\} \cup$
$\{(\mathrm{L}\textsc{ock}\mathrm{E}\ p\ es, l) \mid \mathrm{L}\textsc{ock}\mathrm{E}\ p\ es\ \in\ \mathrm{allL}\ E\ X \wedge l\ \in\ \mathrm{allL}\ E\ X \wedge$
$\quad \exists e.(\mathrm{L\_e}\ l = \mathrm{S}\textsc{ome}\ \ e) \wedge e\ \in\ es \wedge e\ \in\ \mathrm{mem\_accesses}\ E\} \cup$
$\{(l, \mathrm{U}\textsc{nlock}\mathrm{E}\ p\ es) \mid l\ \in\ \mathrm{memL}\ E\ X \wedge \mathrm{U}\textsc{nlock}\mathrm{E}\ p\ es\ \in\ \mathrm{allL}\ E\ X \wedge$
$\quad \exists e.(\mathrm{L\_e}\ l = \mathrm{S}\textsc{ome}\ \ e) \wedge e\ \in\ es\} \cup$
$\{(\mathrm{U}\textsc{nlock}\mathrm{E}\ p\ es, l) \mid \mathrm{U}\textsc{nlock}\mathrm{E}\ p\ es\ \in\ \mathrm{allL}\ E\ X \wedge l\ \in\ \mathrm{allL}\ E\ X \wedge$
$\quad \exists e\ e'.(\mathrm{L\_e}\ l = \mathrm{S}\textsc{ome}\ \ e) \wedge e'\ \in\ es \wedge e\ \notin\ es \wedge$
$\qquad\qquad (e', e) \in\ X.\mathit{memory\_order} \wedge$
$\qquad\qquad (l\ \in\ \mathrm{memL}\ E\ X \vee (e\ \in\ \mathrm{mem\_accesses}\ E \wedge (\mathsf{proc}\ e = p)))\} \cup$
$\{(l, \mathrm{L}\textsc{ock}\mathrm{E}\ p\ es) \mid l\ \in\ \mathrm{memL}\ E\ X \wedge \mathrm{L}\textsc{ock}\mathrm{E}\ p\ es\ \in\ \mathrm{allL}\ E\ X \wedge$
$\quad \exists e\ e'.(\mathrm{L\_e}\ l = \mathrm{S}\textsc{ome}\ \ e) \wedge e'\ \in\ es \wedge e\ \notin\ es \wedge$
$\qquad\qquad (e, e') \in\ X.\mathit{memory\_order}\} \cup$
$\{(\mathrm{U}\textsc{nlock}\mathrm{E}\ p\ es, \mathrm{L}\textsc{ock}\mathrm{E}\ p'\ es') \mid \mathrm{U}\textsc{nlock}\mathrm{E}\ p\ es\ \in\ \mathrm{allL}\ E\ X \wedge \mathrm{L}\textsc{ock}\mathrm{E}\ p'\ es'\ \in\ \mathrm{allL}\ E\ X \wedge$
$\quad es \neq es' \wedge \exists e\ e'.e\ \in\ es \wedge e'\ \in\ es' \wedge (e, e') \in\ X.\mathit{memory\_order}\} \cup$
$\{(\mathrm{WE}\textsc{vt}\ e, \mathrm{WE}\textsc{vt}\ e') \mid \mathrm{WE}\textsc{vt}\ e\ \in\ \mathrm{allL}\ E\ X \wedge \mathrm{WE}\textsc{vt}\ e'\ \in\ \mathrm{allL}\ E\ X \wedge$
$\quad (e, e') \in\ X.\mathit{memory\_order} \wedge (\mathsf{proc}\ e = \mathsf{proc}\ e')\}$

Theorem 22

$\forall E\ X.$
well_formed_event_structure E $\wedge$ linear_valid_execution E X
$\Longrightarrow$
partial_order (lo E X)(allL E X)

Theorem 23

$\forall E\ X.$
well_formed_event_structure E $\wedge$ linear_valid_execution E X
$\Longrightarrow$
label_order E X $\subseteq$ lo E X

Part IX

executable_checker

type_abbrev ch_reln : $('a\#'a)$list

ch_event_structure =⟨| *ch_procs* : proc list;
ch_events : ($'reg$ event)list;
ch_intra_causality : ($'reg$ event)ch_reln;
ch_atomicity : ($'reg$ event)list list|⟩

ch_execution_witness =⟨| (* the memory order is the transitive closure of the pairs in ch_memory_order *)
ch_memory_order : ($'reg$ event)ch_reln;
ch_rfmap : $'reg$ event $\rightarrow$ $'reg$ event option;
ch_initial_state : $'reg$ location $\rightarrow$ value option|⟩

subsetL $r1\ r2 = \forall x\ y.\ \mathrm{MEM}\ (x,y)r1 \implies \mathrm{MEM}\ (x,y)r2$

$(\mathrm{cross}[\,]\_ = [\,]) \wedge$
$(\mathrm{cross}\ ((x,y) \in r)r' = \mathrm{MAP}\ (\lambda(x',y').(x,y'))r' \mathbin{+\!\!+} \mathrm{cross}\ r\ r')$

tinsert $(x,y)r =$
let *left* = FILTER $(\lambda(x',y').y' = x)r$ **in**
let *right* = FILTER $(\lambda(x',y').x' = y)r$ **in**
$(x,y) \in r \mathbin{+\!\!+}$
MAP $(\lambda(x',y').(x',y))$*left* ++
MAP $(\lambda(x',y').(x,y'))$*right* ++
cross *left right*

$(\mathrm{tclose}[\,]acc = acc) \wedge$
$(\mathrm{tclose}\ ((x,y) \in r)acc = \mathrm{tclose}\ r\ (\mathrm{tinsert}\ (x,y)acc))$

transitiveL $r =$
$\forall x\ y\ z.\ \mathrm{MEM}\ (x,y)r \wedge \mathrm{MEM}\ (y,z)r \implies \mathrm{MEM}\ (x,z)r$

cis_mem_access $e =$
case $e.action$ **of**
  ACCESS d (LOCATION_MEM a)$v \rightarrow$ **T**
 $\|$ _ $\rightarrow$ **F**

is_mem_write $e =$
case $e.action$ **of**
  ACCESS W (LOCATION_MEM a)$v \rightarrow$ **T**
 $\|$ _ $\rightarrow$ **F**

is_mem_read $e =$
case $e.action$ **of**
  ACCESS R (LOCATION_MEM a)$v \rightarrow$ **T**

```
 || _ → F
```

```
is_write  e =
case  e.action  of
   Access W l v → T
 || _ → F
```

```
is_read  e =
case  e.action  of
   Access R l v → T
 || _ → F
```

```
is_reg_write  e =
case  e.action  of
   Access W (Location_reg p x)v → T
 || _ → F
```

```
is_reg_read  e =
case  e.action  of
   Access R (Location_reg p x)v → T
 || _ → F
```

```
is_barrier  e =
case  e.action  of
   Barrier Mfence → T
 || _ → F
```

```
check_po_iico  intra e_1 e_2 =
if  proc e_1 = proc e_2  then
   if  e_1.iiid.poi < e_2.iiid.poi  then
      T
    else if  e_1.iiid.poi = e_2.iiid.poi  then
      $e_1 \neq e_2 \wedge$ MEM $(e_1, e_2)$intra
    else
      F
 else
   F
```

```
check_po_iico_in_mo  intra mo e_1 e_2 =
if  check_po_iico  intra e_1 e_2  then
   MEM $(e_1, e_2)$mo
 else
   T
```

```
barrier_separated  intra barriers e_1 e_2 =
(proc $e_1$ = proc $e_2$) $\wedge$
 EXISTS ($\lambda eb.$ check_po_iico $intra\ e_1\ eb$ $\wedge$ check_po_iico $intra\ eb\ e_2$)
        $barriers$


previous_writes1 $er\ r$ =
 MAP FST (FILTER ($\lambda(ew, er').(er' = er) \wedge$ is_write $ew$ $\wedge$ (loc $ew$ = loc $er$))$r$)


previous_writes2 $er\ intra\ es$ =
 FILTER ($\lambda ew.$(loc $ew$ = loc $er$) $\wedge$ check_po_iico $intra\ ew\ er$)$es$


check_maximal1 $x\ xs\ r$ =
 MEM $x\ xs$ $\wedge$
 EVERY ($\lambda x'.$if $x \neq x'$ then $\neg$(MEM $(x, x')r$) else T)$xs$


check_maximal2 $x\ xs\ intra$ =
 MEM $x\ xs$ $\wedge$
 EVERY ($\lambda x'.$if $x \neq x'$ then $\neg$(check_po_iico $intra\ x\ x'$) else T)$xs$


check_valid_execution $E\ X$ =
let $mo$ = tclose (FILTER ($\lambda(e_1, e_2).e_1 \neq e_2$)$X.ch\_memory\_order$)[] in
let writes = FILTER is_write $E.ch\_events$ in
let reads = FILTER is_read $E.ch\_events$ in
let $mwrites$ = FILTER is_mem_write writes in
let $mreads$ = FILTER is_mem_read reads in
let $barriers$ = FILTER is_barrier $E.ch\_events$ in
let $intra$ = tclose $E.ch\_intra\_causality$[] in
(* partial order *)
 EVERY ($\lambda(e_1, e_2).e_1 \neq e_2$)$mo$ $\wedge$
 EVERY ($\lambda(e_1, e_2).cis\_mem\_access\ e_1 \wedge cis\_mem\_access\ e_2 \wedge$
                        MEM $e_1$ $E.ch\_events$ $\wedge$ MEM $e_2$ $E.ch\_events$)$X.ch\_memory\_order$ $\wedge$
(* linear order on mwrites *)
 EVERY ($\lambda e_1.$ EVERY ($\lambda e_2.$if $e_1 \neq e_2$ then
                                   MEM $(e_1, e_2)mo$ $\vee$ MEM $(e_2, e_1)mo$
                                 else
                                   T)
                             $mwrites$)
        $mwrites$ $\wedge$
(* po_iico in memory_order *)
 EVERY ($\lambda er_1.$ EVERY ($\lambda er_2.$ check_po_iico_in_mo $intra\ mo\ er_1\ er_2$)$mreads$)
        $mreads$ $\wedge$
 EVERY ($\lambda er.$ EVERY ($\lambda ew.$ check_po_iico_in_mo $intra\ mo\ er\ ew$)$mwrites$)
        $mreads$ $\wedge$
 EVERY ($\lambda ew_1.$ EVERY ($\lambda ew_2.$ check_po_iico_in_mo $intra\ mo\ ew_1\ ew_2$)$mwrites$)
        $mwrites$ $\wedge$
 EVERY ($\lambda ew.$
   EVERY ($\lambda er.$
    if barrier_separated $intra$ $barriers$ $ew$ $er$ $\lor$
        EXISTS ($\lambda es.$ MEM $ew$ $es$ $\lor$ MEM $er$ $es$)$E.ch\_atomicity$ then
      check_po_iico_in_mo $intra$ $mo$ $ew$ $er$
     else
      T)
    $mreads$)
  $mwrites$ $\land$
(* atomicity *)
EVERY ($\lambda es.$
  EVERY ($\lambda e.$
    if $\neg$(MEM $e$ $es$) then
      EVERY ($\lambda e'.$
        if is_mem_read $e'$ $\lor$ is_mem_write $e'$ then
          MEM $(e, e')mo$
         else
          T)
        $es$ $\lor$
      EVERY ($\lambda e'.$
        if is_mem_read $e'$ $\lor$ is_mem_write $e'$ then
          MEM $(e', e)mo$
         else
          T)
        $es$
     else
      T)
    ($mreads$ ++$mwrites$))
  $E.ch\_atomicity$ $\land$
  (* rfmc *)
  EVERY ($\lambda er.$
    case $X.ch\_rfmap$ $er$ of
      SOME $ew$ $\to$
        is_read $er$ $\land$ is_write $ew$ $\land$ MEM $ew$ $E.ch\_events$ $\land$
        (loc $er$ = loc $ew$) $\land$ (value_of $er$ = value_of $ew$)
    || NONE $\to$ T)
    $E.ch\_events$ $\land$
  (* rfmap written and initial*)
  EVERY ($\lambda er.$
    case $X.ch\_rfmap$ $er$ of
      SOME $ew$ $\to$
        if is_mem_write $ew$ then
          check_maximal1 $ew$ (previous_writes1 $er$ $mo$ ++
                                  previous_writes2 $er$ $intra$ writes)$mo$
         else
          check_maximal2 $ew$ (previous_writes2 $er$ $intra$ writes)$intra$
    || NONE $\to$
        (case loc $er$ of
            SOME $l$ $\to$
              (value_of $er$ = $X.ch\_initial\_state$ $l$) $\land$
              (previous_writes1 $er$ $mo$ = [ ]) $\land$
```

```
                (previous_writes2 $er$ $intra$ writes $= [\,]$)
            $\|$ NONE $\rightarrow$ **F**))
    reads
```

check_set_eq es_1 es_2 = EVERY $(\lambda e.\,$MEM $e\ es_2)es_1 \wedge$ EVERY $(\lambda e.\,$MEM $e\ es_1)es_2$

```
check_well_formed_event_structure $E$ =
let $intra$ = tclose (FILTER $(\lambda(e_1, e_2).e_1 \neq e_2)E.ch\_intra\_causality)[\,]$ in
 EVERY $(\lambda e.\,$MEM (proc $e)E.ch\_procs)E.ch\_events \wedge$
 EVERY $(\lambda e_1.$
   EVERY $(\lambda e_2.$
     if $(e_1.iiid = e_2.iiid) \wedge (e_1.$eiid $= e_2.$eiid$)$ then
       $e_1 = e_2$
     else T)
     $E.ch\_events)$
   $E.ch\_events \wedge$
 EVERY $(\lambda(e_1, e_2).\,$MEM $e_1\ E.ch\_events \wedge$ MEM $e_2\ E.ch\_events)E.ch\_intra\_causality \wedge$
 EVERY $(\lambda(e_1, e_2).e_1 \neq e_2)intra \wedge$
 EVERY $(\lambda(e_1, e_2).e_1.iiid = e_2.iiid)intra \wedge$
$\neg$ MEM$[\,]E.ch\_atomicity \wedge$
 EVERY $(\lambda es.\,$EVERY $(\lambda e.\,$MEM $e\ E.ch\_events)es)E.ch\_atomicity \wedge$
 EVERY $(\lambda es_1.$
   EVERY $(\lambda es_2.$
     if $\neg$ check_set_eq $es_1\ es_2$ then
       EVERY $(\lambda e_1.\,$EVERY $(\lambda e_2.e_1 \neq e_2)es_1)es_2$
     else
       T)
     $E.ch\_atomicity)$
   $E.ch\_atomicity \wedge$
 EVERY $(\lambda es_1.\,$EVERY $(\lambda e_1.\,$EVERY $(\lambda e_2.e_1.iiid = e_2.iiid)es_1)es_1)E.ch\_atomicity \wedge$
 EVERY $(\lambda e.$case loc $e$ of SOME (LOCATION_REG $p\ r) \rightarrow p =$ proc $e \,\|\, \_ \rightarrow$ T$)E.ch\_events \wedge$
 EVERY $(\lambda(e_1, e_2).\neg$ is_mem_write $e_1)intra \wedge$
 EVERY $(\lambda e_1.$
   EVERY $(\lambda e_2.$
     if is_write $e_1 \wedge e_1 \neq e_2 \wedge$ (is_write $e_2 \vee$ is_read $e_2) \wedge$
          $(e_1.iiid = e_2.iiid) \wedge$ (loc $e_1$ = loc $e_2$) then
       MEM $(e_1, e_2)intra \vee$ MEM $(e_2, e_1)intra$
     else
       T)
     $E.ch\_events)$
   $E.ch\_events \wedge$
 EVERY $(\lambda es.$
   EVERY $(\lambda e_1.$
     EVERY $(\lambda e_2.$
       if $e_1.iiid = e_2.iiid$ then MEM $e_2\ es$ else T)
       $E.ch\_events)$
     $es)$
   $E.ch\_atomicity \wedge$
```

$\text{EVERY}\ (\lambda es.\,\text{EXISTS}\ (\lambda e.\,\text{is\_mem\_read}\ \ e)\,es)\,E.\mathit{ch\_atomicity}$

$\text{chE\_to\_E}\ \ E =$
$\langle\!|\ \mathit{procs} := \mathsf{set}\ E.\mathit{ch\_procs};$
$\quad \mathit{events} := \mathsf{set}\ E.\mathit{ch\_events};$
$\quad \mathit{intra\_causality} :=$
$\quad (\mathsf{set}\ E.\mathit{ch\_intra\_causality})^{+} \cup \{(e,e) \mid e\ \in\ (\mathsf{set}\ E.\mathit{ch\_events})\};$
$\quad \mathit{atomicity} := \mathsf{set}\ (\text{MAP}\ \ \mathsf{set}\ E.\mathit{ch\_atomicity})|\!\rangle$

$\text{chX\_to\_X}\ \ E\ X =$
$\langle\!|\ \mathit{memory\_order} := (\mathsf{set}\ X.\mathit{ch\_memory\_order})^{+} \cup$
$\qquad\qquad \{(e,e) \mid e\ \in\ \text{mem\_accesses}\ (\text{chE\_to\_E}\ \ E)\};$
$\quad \mathit{rfmap} := \{(\mathit{ew},\mathit{er}) \mid \text{MEM}\ \ \mathit{er}\ E.\mathit{ch\_events} \wedge (X.\mathit{ch\_rfmap}\ \mathit{er} = \textsc{Some}\ \ \mathit{ew})\};$
$\quad \mathit{initial\_state} := X.\mathit{ch\_initial\_state}|\!\rangle$

Theorem 24

$\forall E\ X.$
$\text{well\_formed\_event\_structure}\ (\text{chE\_to\_E}\ \ E)$
$\Longrightarrow$
$(\text{check\_valid\_execution}\ \ E\ X = \text{valid\_execution}\ (\text{chE\_to\_E}\ \ E)(\text{chX\_to\_X}\ \ E\ X))$

Theorem 25

$\forall E.\,\text{check\_well\_formed\_event\_structure}\ \ E = \text{well\_formed\_event\_structure}\ (\text{chE\_to\_E}\ \ E)$

Part X

correct_typesetting

Theorem 26

typesetting $*previous_writes* =
axiomatic_memory_model $*previous_writes*

Theorem 27

typesetting $*check_rfmap_written* =
axiomatic_memory_model $*check_rfmap_written*

Theorem 28

typesetting $*check_rfmap_initial* =
axiomatic_memory_model $*check_rfmap_initial*

Theorem 29

$\text{valid\_execution}\ E\ X =$
$\text{ve1}\ E\ X \wedge$
$\text{ve2}\ E\ X \wedge$
$\text{ve3}\ E\ X \wedge$
$\text{ve4}\ E\ X \wedge$
$\text{ve5}\ E\ X \wedge$
$\text{ve6}\ E\ X \wedge$
$\text{ve7}\ E\ X \wedge$
$\text{ve8}\ E\ X \wedge$
$\text{ve9}\ E\ X \wedge$
$\text{ve10}\ E\ X \wedge$
$\text{check\_rfmap\_written}\ E\ X \wedge$
$\text{check\_rfmap\_initial}\ E\ X$

Theorem 30

$\forall s\ l\ s'\ x\ y\ z.\, \text{machine\_trans}\ s\ l\ s' = x/*x, z*/s \xrightarrow{l} s'$

Index