Mathematizing C++ Concurrency: The Isabelle/HOL model

Mark Batty Scott Owens Susmit Sarkar Peter Sewell Tjark Weber

University of Cambridge

`http://www.cl.cam.ac.uk/users/pes20/weakmemory`

1. Introduction

This document collects together the Isabelle/HOL definitions of our C++ memory model, in their logical order. For explanation and discussion, please see the associated paper.

The X_{opsem} part of a candidate execution X consists of a

- $\mathit{threadids}$, a set of thread
- $\mathit{actions}$, a set of actions
- *location-kind*, a location typing

and four binary relations over its actions:

- *sequenced-before* (sb),
- *additional-synchronized-with* (asw),
- *data-dependency* (dd), and
- *control-dependency* (cd).

The X_{witness} part of a candidate execution X consists of a further three binary relations over its actions:

- *rf*,
- *sc*, and
- *modification-order* (mo).

Given a candidate execution $X = (X_{\text{opsem}}, X_{\text{witness}})$, this document defines various derived relations:

- *release-sequence*
- *hypothetical-release-sequence*
- *carries-a-dependency-to*
- *dependency-ordered-before*
- *inter-thread-happens-before*
- *happens-before*
- *visible-side-effect*
- vsse

together with the predicates required to define

- $\mathit{consistent\_execution}$

and

- *unsequenced-race*,
- dr, and
- ir.

In the Isabelle/HOL source each definition is explicitly parameterised on the components of a candidate execution and the required derived relations, but here we suppress that parameterisation to reduce clutter. The Isabelle/HOL also contains set-typed versions of some of the predicates, for use in code extraction; we suppress those here also.

2. Auxiliary definitions

$\text{relation\_over}\ s\ \mathit{rel} = \text{domain}\ \mathit{rel} \subseteq s \wedge \text{range}\ \mathit{rel} \subseteq s$

$\xrightarrow{\mathit{rel}}|_s = \mathit{rel} \cap (s \times s)$

$\text{strict\_preorder}\ \mathit{ord} = \text{irreflexive}\ \mathit{ord} \wedge \text{trans}\ \mathit{ord}$

$\text{total\_over}\ s\ \mathit{ord} =$
$\quad \text{relation\_over}\ s\ \mathit{ord} \wedge$
$\quad (\forall x \in s.\ \forall y \in s.\ x \xrightarrow{\mathit{ord}} y \vee y \xrightarrow{\mathit{ord}} x \vee (x = y))$

$\text{strict\_total\_order\_over}\ s\ \mathit{ord} =$
$\quad \text{strict\_preorder}\ \mathit{ord} \wedge \text{total\_over}\ s\ \mathit{ord}$

$x \xmapsto{\mathit{ord}}_{\mathit{pred}} y =$
$\quad \mathit{pred}\ x \wedge x \xrightarrow{\mathit{ord}} y \wedge \neg(\exists z.\ \mathit{pred}\ z \wedge x \xrightarrow{\mathit{ord}} z \xrightarrow{\mathit{ord}} y)$

$x \xmapsto{\mathit{ord}} y =$
$\quad x \xrightarrow{\mathit{ord}} y \wedge \neg(\exists z.\ x \xrightarrow{\mathit{ord}} z \xrightarrow{\mathit{ord}} y)$

3. Types

type_abbrev action_id : string

type_abbrev thread_id : string

type_abbrev location : string

type_abbrev val : string

location_kind =
  MUTEX
 | NON_ATOMIC
 | ATOMIC

memory_order_enum =
  MO_SEQ_CST
 | MO_RELAXED
 | MO_RELEASE
 | MO_ACQUIRE
 | MO_CONSUME
 | MO_ACQ_REL

action =
  LOCK **of** action_id thread_id location
 | UNLOCK **of** action_id thread_id location
 | ATOMIC_LOAD **of** action_id thread_id memory_order_enum location val
 | ATOMIC_STORE **of** action_id thread_id memory_order_enum location val
 | ATOMIC_RMW **of** action_id thread_id memory_order_enum location val val
 | LOAD **of** action_id thread_id location val
 | STORE **of** action_id thread_id location val
 | FENCE **of** action_id thread_id memory_order_enum

4. Auxiliary functions over actions

(action_id_of (LOCK *aid* _ _) = *aid*) ∧
(action_id_of (UNLOCK *aid* _ _) = *aid*) ∧
(action_id_of (ATOMIC_LOAD *aid* _ _ _ _) = *aid*) ∧
(action_id_of (ATOMIC_STORE *aid* _ _ _ _) = *aid*) ∧
(action_id_of (ATOMIC_RMW *aid* _ _ _ _ _) = *aid*) ∧
(action_id_of (LOAD *aid* _ _ _) = *aid*) ∧
(action_id_of (STORE *aid* _ _ _) = *aid*) ∧
(action_id_of (FENCE *aid* _ _) = *aid*)

(thread_id_of (LOCK _ *tid* _) = *tid*) ∧
(thread_id_of (UNLOCK _ *tid* _) = *tid*) ∧
(thread_id_of (ATOMIC_LOAD _ *tid* _ _ _) = *tid*) ∧
(thread_id_of (ATOMIC_STORE _ *tid* _ _ _) = *tid*) ∧
(thread_id_of (ATOMIC_RMW _ *tid* _ _ _ _) = *tid*) ∧
(thread_id_of (LOAD _ *tid* _ _) = *tid*) ∧
(thread_id_of (STORE _ *tid* _ _) = *tid*) ∧
(thread_id_of (FENCE _ *tid* _) = *tid*)

same_thread *a b* = (thread_id_of *a* = thread_id_of *b*)

threadwise_relation_over *s rel* =
  relation_over *s rel* ∧ (∀(*a*, *b*) ∈ *rel*. same_thread *a b*)

(location (LOCK _ _ *l*) = SOME *l*) ∧
(location (UNLOCK _ _ *l*) = SOME *l*) ∧
(location (ATOMIC_LOAD _ _ _ *l* _) = SOME *l*) ∧
(location (ATOMIC_STORE _ _ _ *l* _) = SOME *l*) ∧
(location (ATOMIC_RMW _ _ _ *l* _ _) = SOME *l*) ∧
(location (LOAD _ _ *l* _) = SOME *l*) ∧
(location (STORE _ _ *l* _) = SOME *l*) ∧
(location (FENCE _ _ _) = NONE)

same_location *a b* = (location *a* = location *b*)

(value_read (ATOMIC_LOAD _ _ _ _ *v*) = SOME *v*) ∧
(value_read (ATOMIC_RMW _ _ _ _ *v* _) = SOME *v*) ∧
(value_read (LOAD _ _ _ *v*) = SOME *v*) ∧
(value_read _ = NONE)

(value_written (ATOMIC_STORE _ _ _ _ *v*) = SOME *v*) ∧
(value_written (ATOMIC_RMW _ _ _ _ _ *v*) = SOME *v*) ∧
(value_written (STORE _ _ _ *v*) = SOME *v*) ∧
(value_written _ = NONE)

(memory_order (ATOMIC_LOAD _ _ *modification-order* _ _) =
  SOME *modification-order*) ∧
(memory_order (ATOMIC_STORE _ _ *modification-order* _ _) =
  SOME *modification-order*) ∧
(memory_order (ATOMIC_RMW _ _ *modification-order* _ _ _) =
  SOME *modification-order*) ∧
(memory_order (FENCE _ _ *modification-order*) =
  SOME *modification-order*) ∧
(memory_order _ =
  NONE)

is_lock *a* =
  case *a* **of** LOCK _ _ _ → **T** ∥ _ → **F**

is_unlock *a* =
  case *a* **of** UNLOCK _ _ _ → **T** ∥ _ → **F**

is_atomic_load *a* =
  case *a* **of** ATOMIC_LOAD _ _ _ _ _ → **T** ∥ _ → **F**

is_atomic_store *a* =
  case *a* **of** ATOMIC_STORE _ _ _ _ _ → **T** ∥ _ → **F**

is_atomic_rmw *a* =
  case *a* **of** ATOMIC_RMW _ _ _ _ _ _ → **T** ∥ _ → **F**

is_load *a* = **case** *a* **of** LOAD _ _ _ _ → **T** ∥ _ → **F**

is_store *a* = **case** *a* **of** STORE _ _ _ _ → **T** ∥ _ → **F**

is_fence *a* = **case** *a* **of** FENCE _ _ _ → **T** ∥ _ → **F**

is_lock_or_unlock *a* = is_lock *a* ∨ is_unlock *a*

is_atomic_action *a* =
  is_atomic_load *a* ∨ is_atomic_store *a* ∨ is_atomic_rmw *a*

is_load_or_store *a* = is_load *a* ∨ is_store *a*

is_synchronization_action *a* =
  is_lock_or_unlock *a* ∨ is_atomic_action *a*

is_read *a* =
  is_atomic_load *a* ∨ is_atomic_rmw *a* ∨ is_load *a*

is_write *a* =
  is_atomic_store *a* ∨ is_atomic_rmw *a* ∨ is_store *a*

is_acquire *a* =
  (**case** memory_order *a* **of**
    SOME *modification-order* →
      modification-order ∈
        {MO_ACQUIRE, MO_ACQ_REL, MO_SEQ_CST} ∧
      (is_read *a* ∨ is_fence *a*) ∨
      (* 29.8:5 states that consume fences are acquire fences. *)
      (*modification-order* = MO_CONSUME) ∧ is_fence *a*
  ∥ NONE → is_lock *a*)

is_consume *a* =
  is_read *a* ∧ (memory_order *a* = SOME MO_CONSUME)

is_release *a* =
  (**case** memory_order *a* **of**
    SOME *modification-order* →
      modification-order ∈ {MO_RELEASE, MO_ACQ_REL, MO_SEQ_CST} ∧
    (is_write *a* ∨ is_fence *a*)
  ∥ NONE → is_unlock *a*)

is_seq_cst *a* = (memory_order *a* = SOME MO_SEQ_CST)

5. Well-formed threads

well_formed_action *a* =
  case *a* **of**
    ATOMIC_LOAD _ _ *modification-order* _ _ → *modification-order* ∈
      {MO_RELAXED, MO_ACQUIRE, MO_SEQ_CST, MO_CONSUME}
  ∥ ATOMIC_STORE _ _ *modification-order* _ _ → *modification-order* ∈
      {MO_RELAXED, MO_RELEASE, MO_SEQ_CST}
  ∥ ATOMIC_RMW _ _ *modification-order* _ _ _ → *modification-order* ∈
      {MO_RELAXED, MO_RELEASE, MO_ACQUIRE, MO_ACQ_REL,
        MO_SEQ_CST, MO_CONSUME}
  ∥ _ → **T**

locations_of *actions* = {*l*. ∃*a*. (location *a* = SOME *l*)}

actions_respect_location_kinds =
  $\forall a.$
    case location a **of** SOME $l \rightarrow$
      (**case** *location-kind* l **of**
        MUTEX $\rightarrow$ is_lock_or_unlock a
      $\|$ NON_ATOMIC $\rightarrow$ is_load_or_store a
      $\|$ ATOMIC $\rightarrow$ is_load_or_store $a \vee$ is_atomic_action a)
    $\|$ NONE $\rightarrow$ **T**

is_at_location_kind = is_at_location_kind =
  case location a **of**
    SOME $l \rightarrow$ (*location-kind* l = *lk0*)
  $\|$ NONE $\rightarrow$ **F**

is_at_mutex_location a =
  is_at_location_kind a MUTEX

is_at_non_atomic_location a =
  is_at_location_kind a NON_ATOMIC

is_at_atomic_location a =
  is_at_location_kind a ATOMIC

well_formed_threads =
  inj_on action_id_of ($actions$) $\wedge$
  ($\forall a.$ well_formed_action a) $\wedge$
  threadwise_relation_over $actions$ *sequenced-before* $\wedge$
  threadwise_relation_over $actions$ *data-dependency* $\wedge$
  threadwise_relation_over $actions$ *control-dependency* $\wedge$
  strict_preorder *sequenced-before* $\wedge$
  strict_preorder *data-dependency* $\wedge$
  strict_preorder *control-dependency* $\wedge$
  relation_over $actions$ *additional-synchronized-with* $\wedge$
  ($\forall a.$ thread_id_of $a \in$ threads) $\wedge$
  actions_respect_location_kinds $\wedge$
  data-dependency $\subseteq$ *sequenced-before*

6. Consistent locks

all_lock_or_unlock_actions_at $lopt\ as$ =
  $\{a \in as.$ is_lock_or_unlock $a \wedge$ (location $a = lopt$)$\}$

consistent_locks =
$\forall l \in$ locations_of $actions$. (*location-kind* l = MUTEX) $\Longrightarrow$ (
  let $lock\_unlock\_actions$ =
    all_lock_or_unlock_actions_at (SOME l)$actions$ **in**
  let $lock\_order = \xrightarrow{sc}|_{lock\_unlock\_actions}$ **in**
  (* 30.4.1:5 - The implementation shall serialize those (lock and unlock) operations. *)
  strict_total_order_over $lock\_unlock\_actions\ lock\_order$ $\wedge$

  (* 30.4.1:1 A thread owns a mutex from the time it successfully calls one of the lock functions until it calls unlock.*)
  (* 30.4.1:20 Requires: The calling thread shall own the mutex. *)
  (* 30.4.1:21 Effects: Releases the calling threads ownership of the mutex.*)
  ($\forall a_u \in lock\_unlock\_actions.$ is_unlock $a_u \Longrightarrow$
    ($\exists a_l \in lock\_unlock\_actions.$
      $a_l \xmapsto{lock\_order} a_u$ $\wedge$ same_thread $a_l\ a_u$ $\wedge$ is_lock a_l)) $\wedge$

  (* 30.4.1:7 Effects: Blocks the calling thread until ownership of the mutex can be obtained for the calling thread.*)
  (* 30.4.1:8 Postcondition: The calling thread owns the mutex. *)
  ($\forall a_l \in lock\_unlock\_actions.$ is_lock $a_l \Longrightarrow$
    ($\forall a_u \in lock\_unlock\_actions.$
      $a_u \xmapsto{lock\_order} a_l \Longrightarrow$ is_unlock a_u)))

7. Release sequences

rs_element $rs\_head\ a$ =
  same_thread $a\ rs\_head$ $\vee$ is_atomic_rmw a

$a_{rel} \xrightarrow{release\text{-}sequence} b$ =
  is_at_atomic_location b $\wedge$
  is_release a_{rel} $\wedge$ (
    ($b = a_{rel}$) $\vee$
    (rs_element $a_{rel}\ b \wedge a_{rel} \xrightarrow{modification\text{-}order} b$ $\wedge$
      ($\forall c.\ a_{rel} \xrightarrow{modification\text{-}order} c \xrightarrow{modification\text{-}order} b \Longrightarrow$
        rs_element $a_{rel}\ c$)))

$a \xrightarrow{hypothetical\text{-}release\text{-}sequence} b$ =
  is_at_atomic_location b $\wedge$ (
  ($b = a$) $\vee$
  (rs_element $a\ b \wedge a \xrightarrow{modification\text{-}order} b$ $\wedge$
    ($\forall c.\ a \xrightarrow{modification\text{-}order} c \xrightarrow{modification\text{-}order} b \Longrightarrow$
      rs_element $a\ c$)))

8. Synchronizes-with

$a \xrightarrow{synchronizes\text{-}with} b$ =
  (* – additional synchronization, from thread create etc. – *)
  $a \xrightarrow{additional\text{-}synchronized\text{-}with} b$ $\vee$

  (same_location $a\ b \wedge a \in actions \wedge b \in actions$ $\wedge$ (
    (* – mutex synchronization – *)
    (is_unlock a $\wedge$ is_lock $b \wedge a \xrightarrow{sc} b$) $\vee$

    (* – release/acquire synchronization – *)
    (is_release a $\wedge$ is_acquire b $\wedge \neg$ same_thread $a\ b$ $\wedge$
    ($\exists c.\ a \xrightarrow{release\text{-}sequence} c \xrightarrow{rf} b$)) $\vee$

    (* – fence synchronization – *)
    (is_fence a $\wedge$ is_release a $\wedge$ is_fence b $\wedge$ is_acquire b $\wedge$
    ($\exists x.\ \exists y.$ same_location $x\ y$ $\wedge$
      is_atomic_action x $\wedge$ is_atomic_action y $\wedge$ is_write x $\wedge$
      $a \xrightarrow{sequenced\text{-}before} x \wedge y \xrightarrow{sequenced\text{-}before} b$ $\wedge$
      ($\exists z.\ x \xrightarrow{hypothetical\text{-}release\text{-}sequence} z \xrightarrow{rf} y$))) $\vee$

    (is_fence a $\wedge$ is_release a $\wedge$
    is_atomic_action b $\wedge$ is_acquire b $\wedge$
    ($\exists x.$ same_location $x\ b$ $\wedge$
      is_atomic_action x $\wedge$ is_write x $\wedge$
      $a \xrightarrow{sequenced\text{-}before} x$ $\wedge$
      ($\exists z.\ x \xrightarrow{hypothetical\text{-}release\text{-}sequence} z \xrightarrow{rf} b$))) $\vee$

    (is_atomic_action a $\wedge$ is_release a $\wedge$
    is_fence b $\wedge$ is_acquire b $\wedge$
    ($\exists x.$ same_location $a\ x$ $\wedge$ is_atomic_action x $\wedge$
      $x \xrightarrow{sequenced\text{-}before} b$ $\wedge$
      ($\exists z.\ a \xrightarrow{release\text{-}sequence} z \xrightarrow{rf} x$)))))

9. Carries-a-dependency-to

$a \xrightarrow{carries\text{-}a\text{-}dependency\text{-}to} b$ =
  $a\ ((\xrightarrow{rf} \cap \xrightarrow{sequenced\text{-}before}) \cup \xrightarrow{data\text{-}dependency})^+\ b$

10. Dependency-ordered-before

$a \xrightarrow{dependency\text{-}ordered\text{-}before} d$ =
  $a \in actions \wedge d \in actions$ $\wedge$
  ($\exists b.$ is_release a $\wedge$ is_consume b $\wedge$

$(\exists e.\ a \xrightarrow{\textit{release-sequence}} e \xrightarrow{\textit{rf}} b) \land$
$(b \xrightarrow{\textit{carries-a-dependency-to}} d \lor (b = d)))$

11. Inter-thread-happens-before and happens-before

$\xrightarrow{\textit{inter-thread-happens-before}} =$
let $r = \xrightarrow{\textit{synchronizes-with}} \cup$
$\xrightarrow{\textit{dependency-ordered-before}} \cup$
$(\xrightarrow{\textit{synchronizes-with}} \circ \xrightarrow{\textit{sequenced-before}})$ **in**
$(\xrightarrow{r} \cup\ (\xrightarrow{\textit{sequenced-before}} \circ \xrightarrow{r}))^{+}$

consistent_inter_thread_happens_before =
irreflexive $(\xrightarrow{\textit{inter-thread-happens-before}})$

$\xrightarrow{\textit{happens-before}} =$
$\xrightarrow{\textit{sequenced-before}} \cup \xrightarrow{\textit{inter-thread-happens-before}}$

12. Consistent SC order

all_sc_actions =
$\{a.\ (\text{is\_seq\_cst}\ \ a \lor \text{is\_lock}\ \ a \lor \text{is\_unlock}\ \ a)\}$

consistent_sc_order =
let $\text{sc\_happens\_before} = \xrightarrow{\textit{happens-before}}|_{\text{all\_sc\_actions}}$ **in**
let $\text{sc\_mod\_order} = \xrightarrow{\textit{modification-order}}|_{\text{all\_sc\_actions}}$ **in**
$\text{strict\_total\_order\_over all\_sc\_actions}\ (\xrightarrow{sc}) \land$
$\xrightarrow{\text{sc\_happens\_before}} \subseteq \xrightarrow{sc} \land$
$\xrightarrow{\text{sc\_mod\_order}} \subseteq \xrightarrow{sc}$

13. Consistent modification order

consistent_modification_order =
$(\forall a.\ \forall b.\ a \xrightarrow{\textit{modification-order}} b \implies \text{same\_location}\ \ a\ b) \land$
$(\forall l\ \in$ locations_of $\textit{actions}$. **case** *location-kind* l **of**
ATOMIC $\rightarrow$ (
let $\textit{actions\_at\_l} = \{a.\ (\text{location}\ \ a = \text{SOME}\ \ l)\}$ **in**
let $\textit{writes\_at\_l} = \{\textit{a\_at\_l}.\ (\text{is\_store}\ \ a \lor$
$\text{is\_atomic\_store}\ \ a \lor \text{is\_atomic\_rmw}\ \ a)\}$ **in**
strict_total_order_over $\textit{writes\_at\_l}$
$(\xrightarrow{\textit{modification-order}}|_{\textit{actions\_at\_l}}) \land$
(* happens-before at the writes of l is a subset of mo for l *)
$\xrightarrow{\textit{happens-before}}|_{\textit{writes\_at\_l}} \subseteq \xrightarrow{\textit{modification-order}} \land$
(* MO_SEQ_CST fences impose modification order *)
$(\xrightarrow{\textit{sequenced-before}} \circ\ (\xrightarrow{sc}|_{\text{is\_fence}}) \circ \xrightarrow{\textit{sequenced-before}}|_{\textit{writes\_at\_l}})$
$\subseteq \xrightarrow{\textit{modification-order}})$
$\|\ \_ \rightarrow$ (
let $\textit{actions\_at\_l} = \{a.\ (\text{location}\ \ a = \text{SOME}\ \ l)\}$ **in**
$(\xrightarrow{\textit{modification-order}}|_{\textit{actions\_at\_l}}) = \{\}))$

14. Visible side effects and visible sequences of side effects

$a \xrightarrow{\textit{visible-side-effect}} b =$
$a \xrightarrow{\textit{happens-before}} b \land$
$\text{is\_write}\ \ a \land \text{is\_read}\ \ b \land \text{same\_location}\ \ a\ b \land$
$\neg(\exists c.\ (c \neq a) \land (c \neq b) \land$
$\text{is\_write}\ \ c \land \text{same\_location}\ \ c\ b \land$
$a \xrightarrow{\textit{happens-before}} c \xrightarrow{\textit{happens-before}} b)$

visible_sequence_of_side_effects_tail $\textit{vsse\_head}\ b$ =
$\{c.\ \textit{vsse\_head} \xrightarrow{\textit{modification-order}} c\ \land$
$\neg(b \xrightarrow{\textit{happens-before}} c) \land$
$(\forall a.\ \textit{vsse\_head} \xrightarrow{\textit{modification-order}} a \xrightarrow{\textit{modification-order}} c$
$\implies \neg(b \xrightarrow{\textit{happens-before}} a))\}$

visible_sequences_of_side_effects =
$\lambda(\textit{vsse\_head}, b)$.
$(b,$ **if** is_at_atomic_location b **then**
$\{\textit{vsse\_head}\} \cup$
visible_sequence_of_side_effects_tail $\textit{vsse\_head}\ b$
else
$\{\})$

15. Well-formed reads-from mapping

well_formed_reads_from_mapping =
relation_over $\textit{actions}\ (\xrightarrow{rf}) \land$
$(\forall a.\ \forall a'.\ \forall b.\ a \xrightarrow{rf} b \land a' \xrightarrow{rf} b \implies (a = a')) \land$
$(\forall (a, b) \in \xrightarrow{rf}.$
$\text{same\_location}\ \ a\ b \land$
$(\text{value\_read}\ \ b = \text{value\_written}\ \ a) \land$
$(a \neq b) \land$
$(\text{is\_at\_mutex\_location}\ \ a \implies$
$\text{is\_unlock}\ \ a \land \text{is\_lock}\ \ b) \land$
$(\text{is\_at\_non\_atomic\_location}\ \ a \implies$
$\text{is\_store}\ \ a \land \text{is\_load}\ \ b) \land$
$(\text{is\_at\_atomic\_location}\ \ a \implies$
$(\text{is\_atomic\_store}\ \ a \lor \text{is\_atomic\_rmw}\ \ a \lor \text{is\_store}\ \ a)$
$\land (\text{is\_atomic\_load}\ \ b \lor \text{is\_atomic\_rmw}\ \ b \lor \text{is\_load}\ \ b)))$

16. Consistent reads-from mapping

consistent_reads_from_mapping =
$(\forall b.\ (\text{is\_read}\ \ b \land \text{is\_at\_non\_atomic\_location}\ \ b) \implies$
(**if** $(\exists a_{vse}.\ a_{vse} \xrightarrow{\textit{visible-side-effect}} b)$
then $(\exists a_{vse}.\ a_{vse} \xrightarrow{\textit{visible-side-effect}} b \land a_{vse} \xrightarrow{rf} b)$
else $\neg(\exists a.\ a \xrightarrow{rf} b))) \land$

$(\forall b.\ (\text{is\_read}\ \ b \land \text{is\_at\_atomic\_location}\ \ b) \implies$
(**if** $(\exists (b', \textit{vsse}) \in$ *visible-sequences-of-side-effects*. $(b' = b))$
then $(\exists (b', \textit{vsse}) \in$ *visible-sequences-of-side-effects*.
$(b' = b) \land (\exists c\ \in\ \textit{vsse}.\ c \xrightarrow{rf} b))$
else $\neg(\exists a.\ a \xrightarrow{rf} b))) \land$

$(\forall (x, a) \in \xrightarrow{rf}.$
$\forall (y, b) \in \xrightarrow{rf}.$
$a \xrightarrow{\textit{happens-before}} b \land$
$\text{same\_location}\ \ a\ b \land \text{is\_at\_atomic\_location}\ \ b$
$\implies (x = y) \lor x \xrightarrow{\textit{modification-order}} y) \land$

$(\forall (a, b) \in \xrightarrow{rf}.\ \text{is\_atomic\_rmw}\ \ b$
$\implies a \xmapsto{\textit{modification-order}} b) \land$

$(\forall (a, b) \in \xrightarrow{rf}.\ \text{is\_seq\_cst}\ \ b$
$\implies \neg\,\text{is\_seq\_cst}\ \ a \lor$
$a \xmapsto{sc}_{\lambda c.\ \text{is\_write}\ c \land \text{same\_location}\ b\ c} b) \land$

(* -Fence restrictions- *)

(* 29.3:3 *)
$(\forall a.\ \forall (x, b) \in \xrightarrow{\textit{sequenced-before}}.\ \forall y.$
$(\text{is\_fence}\ \ x \land \text{is\_seq\_cst}\ \ x \land \text{is\_atomic\_action}\ \ b \land$

$\text{is\_write}\ a \wedge \text{same\_location}\ a\ b \wedge$
$a \xrightarrow{sc} x \wedge y \xrightarrow{rf} b)$
$\implies (y = a) \vee a \xrightarrow{\textit{modification-order}} y) \wedge$

(* 29.3:4 *)
$(\forall (a, x) \in \xrightarrow{\textit{sequenced-before}}.\ \forall (y, b) \in \xrightarrow{rf}.$
$(\text{is\_atomic\_action}\ a \wedge \text{is\_fence}\ x \wedge \text{is\_seq\_cst}\ x \wedge$
$\text{is\_write}\ a \wedge \text{same\_location}\ a\ b \wedge$
$x \xrightarrow{sc} b \wedge \text{is\_atomic\_action}\ b)$
$\implies (y = a) \vee a \xrightarrow{\textit{modification-order}} y) \wedge$

(* 29.3:5 *)
$(\forall (a, x) \in \xrightarrow{\textit{sequenced-before}}.\ \forall (y, b) \in \xrightarrow{\textit{sequenced-before}}.\ \forall z.$
$(\text{is\_atomic\_action}\ a \wedge \text{is\_fence}\ x \wedge \text{is\_seq\_cst}\ x \wedge$
$\text{is\_write}\ a \wedge \text{is\_fence}\ y \wedge \text{is\_seq\_cst}\ y \wedge$
$\text{is\_atomic\_action}\ b \wedge$
$x \xrightarrow{sc} y \wedge z \xrightarrow{rf} b)$
$\implies (z = a) \vee a \xrightarrow{\textit{modification-order}} z)$

17. All data dependency

$\xrightarrow{all\_data\_dependency} =$
$(\xrightarrow{rf} \cup \xrightarrow{\textit{carries-a-dependency-to}})^+$

18. Consistent executions

$\text{consistent\_execution} =$
$\text{well\_formed\_threads} \wedge$
$\text{consistent\_locks} \wedge$
$\text{consistent\_inter\_thread\_happens\_before} \wedge$
$\text{consistent\_sc\_order} \wedge$
$\text{consistent\_modification\_order} \wedge$
$\text{well\_formed\_reads\_from\_mapping} \wedge$
$\text{consistent\_reads\_from\_mapping}$

19. Sources of undefined behaviour

$\text{indeterminate\_reads} =$
$\{b.\ \text{is\_read}\ b \wedge \neg(\exists a.\ a \xrightarrow{rf} b)\}$

$\text{unsequenced\_races} = \{(a, b).$
$\text{is\_load\_or\_store}\ a \wedge \text{is\_load\_or\_store}\ b \wedge$
$(a \neq b) \wedge \text{same\_location}\ a\ b \wedge (\text{is\_write}\ a \vee \text{is\_write}\ b) \wedge$
$\text{same\_thread}\ a\ b \wedge$
$\neg(a \xrightarrow{\textit{sequenced-before}} b \vee b \xrightarrow{\textit{sequenced-before}} a)\}$

$\text{data\_races} = \{(a, b).$
$(a \neq b) \wedge \text{same\_location}\ a\ b \wedge (\text{is\_write}\ a \vee \text{is\_write}\ b) \wedge$
$\neg\,\text{same\_thread}\ a\ b \wedge$
$\neg(\text{is\_atomic\_action}\ a \wedge \text{is\_atomic\_action}\ b) \wedge$
$\neg(a \xrightarrow{\textit{happens-before}} b \vee b \xrightarrow{\textit{happens-before}} a)\}$

20. C++ memory model

The top-level definition of the memory model is:

$\text{cpp\_memory\_model}\ \textit{opsem}\ (p : \mathsf{program}) =$
let $\text{pre\_executions} = \{(X_{\text{opsem}}, X_{\text{witness}}).$
$\textit{opsem}\ p\ X_{\text{opsem}} \wedge$
$\text{consistent\_execution}\ (X_{\text{opsem}}, X_{\text{witness}})\}$ **in**
if $\exists X \in \text{pre\_executions}\,.$
$(\text{indeterminate\_reads}\ X \neq \{\}) \vee$
$(\text{unsequenced\_races}\ X \neq \{\}) \vee$
$(\text{data\_races}\ X \neq \{\})$
then NONE
else SOME pre_executions