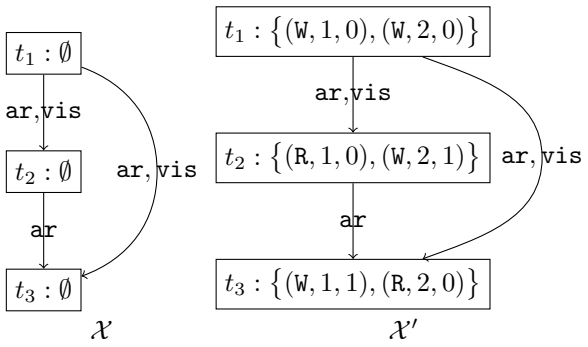

Figure 1: The abstract execution $\mathcal{X}$ is a element of the unit set $\mathbf{1}_{\mathcal{X}}$, and here $\mathcal{X} \bullet_{\mathcal{X}} \mathcal{X}' = \mathcal{X}'$ as the two abstract executions have the same structure the compositions of each nodes exist.

0.1 Program Semantics

To model the global machine states, instead of heap-based states, we use *abstract executions* (Def. 0.1). A *abstract execution* is a graph where each node represents a committed transaction with a unique transaction identifier, and its associated events that have global effect, *i.e.* the first reads and last writes. There are three types of edges in the graph, a *program order* that is a total order for transactions from the same thread, a *visibility relation* that decides the observable history (a set of transactions) for each transaction, and an *arbitration order* that decides the actual global state that is not necessary to be the same as the observable state for each transaction [**? ? ?**].

Definition 0.1 (Runtime abstract executions and abstract executions)**.** Assuming a set of *transactions identifiers* TransID $\triangleq \{t, \ldots\}$, the *transactions* $\mathcal{T} \in$ Tx is defined as a finite partial function from transactions identifiers TransID to valid sets of events Opset,

$$\mathcal{T} \in \textsc{Tx} \quad \triangleq \quad \textsc{TransID} \xrightarrow{fin} \textsc{Opset}$$

An *runtime abstract execution* is a tuple $\hat{\mathcal{X}} = (\mathcal{T}, \mathcal{I}, \hat{\mathtt{po}}, \mathtt{vis}, \mathtt{ar}) \in$ RunAbsExecs that satisfies the following conditions,

- The *transactions* $\mathcal{T}$ is a partial function from transaction identifiers to their corresponding events (Def. **??**).
- Assume a countably infinite set of thread identifiers $i \in \mathcal{I} \subseteq$ ThreadID. The *runtime threads* $\mathcal{I}$ is a set of transaction identifiers.
- The *arbitration order* $\mathtt{ar} \subseteq \mathrm{dom}(\mathcal{T}) \times \mathrm{dom}(\mathcal{T})$ is a strict, total order[1].
- The *runtime program order* $\hat{\mathtt{po}} \subseteq \mathrm{dom}(\mathcal{T}) \times (\mathrm{dom}(\mathcal{T}) \uplus \mathcal{I})$ is the union of several disjoint, strict total orders $\hat{\mathtt{po}}_i$. That is, there exists a partition $\{\mathrm{dom}(\mathcal{T})_i\}_{i \in I}$ of $\mathrm{dom}(\mathcal{T})$ such that $\hat{\mathtt{po}} = \biguplus_{i \in I} \hat{\mathtt{po}}_i$, where $\hat{\mathtt{po}}_i$ is a strict, total order. It also requires $\mathtt{po} \subseteq \mathtt{ar}$.
- The *visibility relation* $\mathtt{vis} \subseteq \mathrm{dom}(\mathcal{T}) \times \mathrm{dom}(\mathcal{T})$ is a relation such that $\mathtt{vis} \subseteq \mathtt{ar}$.

The set of *abstract executions* $\mathcal{X} = (\mathcal{T}, \mathtt{po}, \mathtt{vis}, \mathtt{ar}) \in$ AbsExecs, where the *program order* $\mathtt{po} \subseteq \mathrm{dom}(\mathcal{T}) \times \mathrm{dom}(\mathcal{T})$, is defined by erasing the runtime threads from the runtime abstract executions RunAbsExecs,

$$\textsc{AbsExecs} \quad \triangleq \quad \left\{ [\![\hat{\mathcal{X}}]\!] \;\middle|\; \hat{\mathcal{X}} \in \textsc{RunAbsExecs} \right\}$$

where the erasing function $[\![.]\!] : \textsc{RunAbsExecs} \to \textsc{AbsExecs}$ converts a runtime abstract execution to an abstract execution by erasing the second element, *i.e.* the set of thread identifiers, and also any thread identifiers from the runtime program order,

$$[\![(\mathcal{T}, \mathcal{I}, \hat{\mathtt{po}}, \mathtt{vis}, \mathtt{ar})]\!] \quad \triangleq \quad (\mathcal{T}, \hat{\mathtt{po}} \setminus \{(t, i) \mid t \in \mathrm{dom}(\mathcal{T}) \wedge i \in \mathcal{I}\}, \mathtt{vis}, \mathtt{ar})$$

For brevity, the $\hat{\mathcal{X}} \lfloor_{\mathcal{T}}$, $\hat{\mathcal{X}} \lfloor_{\mathcal{I}}$, $\hat{\mathcal{X}} \lfloor_{\mathtt{po}}$, $\hat{\mathcal{X}} \lfloor_{\mathtt{vis}}$ and $\hat{\mathcal{X}} \lfloor_{\mathtt{ar}}$ denote the corresponding elements in the tuple and

[1]Recall that a relation $R \subseteq A \times A$ is a strict partial order if it is irreflexive and transitive. It is a strict total order if for any $a_1, a_2 \in A$, either $a_1 = a_2$, $(a_1, a_2) \in R$ or $(a_2, a_1) \in R$.

similarly for $\mathcal{X}|_{(.)}$. The composition of two runtime abstract executions, $\bullet_{\hat{\mathcal{X}}} : \textsc{RunAbsExecs} \times \textsc{RunAbsExecs} \rightharpoonup \textsc{RunAbsExecs}$, is defined as the follows,

$$\begin{array}{rcl} \hat{\mathcal{X}}_1 \bullet_{\hat{\mathcal{X}}} \hat{\mathcal{X}}_2 & \triangleq & \begin{cases} \left(\lambda t.\, \hat{\mathcal{X}}_1|_{\mathcal{T}}\ (t) \bullet_o \hat{\mathcal{X}}_2|_{\mathcal{T}}\ (t), \hat{\mathcal{X}}_1|_{\mathcal{I}}, \hat{\mathcal{X}}_1|_{\mathtt{po}}, \hat{\mathcal{X}}_1|_{\mathtt{vis}}, \hat{\mathcal{X}}_1|_{\mathtt{ar}}\right) & \dagger \\ \text{undefined} & \text{otherwise} \end{cases} \\ \dagger & \equiv & \mathrm{dom}(\hat{\mathcal{X}}_1|_{\mathcal{T}}) = \mathrm{dom}(\hat{\mathcal{X}}_2|_{\mathcal{T}}) \wedge \hat{\mathcal{X}}_1|_{\mathcal{I}} = \hat{\mathcal{X}}_2|_{\mathcal{I}} \\ & & \quad \wedge \hat{\mathcal{X}}_1|_{\mathtt{po}} = \hat{\mathcal{X}}_2|_{\mathtt{po}} \wedge \hat{\mathcal{X}}_1|_{\mathtt{vis}} = \hat{\mathcal{X}}_2|_{\mathtt{vis}} \wedge \hat{\mathcal{X}}_1|_{\mathtt{ar}} = \hat{\mathcal{X}}_2|_{\mathtt{ar}} \end{array}$$

The set of the units is $\mathbf{1}_{\mathcal{X}} \triangleq \left\{\mathcal{X} \mid \forall t.\, \mathcal{X}|_{\mathcal{T}}\ (t) = \mathbf{1}_o\right\}$. Then, the order between two runtime abstract executions $\hat{\mathcal{X}}_1$ and $\hat{\mathcal{X}}_2$ is defined as point-wise set inclusions,

$$\begin{array}{rcl} \hat{\mathcal{X}}_1 \sqsubseteq \hat{\mathcal{X}}_2 & \overset{\text{def}}{\iff} & (\forall t.\, \hat{\mathcal{X}}_1|_{\mathcal{T}}\ (t) \implies \hat{\mathcal{X}}_2|_{\mathcal{T}}\ (t)) \wedge \hat{\mathcal{X}}_1|_{\mathcal{I}} \subseteq \hat{\mathcal{X}}_2|_{\mathcal{I}} \\ & & \quad \wedge \hat{\mathcal{X}}_1|_{\mathtt{po}} \subseteq \hat{\mathcal{X}}_2|_{\mathtt{po}} \wedge \hat{\mathcal{X}}_1|_{\mathtt{vis}} \subseteq \hat{\mathcal{X}}_2|_{\mathtt{vis}} \wedge \hat{\mathcal{X}}_1|_{\mathtt{ar}} \subseteq \hat{\mathcal{X}}_2|_{\mathtt{ar}} \end{array}$$

Last, by erasing the runtime, the order between two abstract executions is defined as the follows,

$$\mathcal{X}_1 \sqsubseteq \mathcal{X}_2 \quad \overset{\text{def}}{\iff} \quad \exists \hat{\mathcal{X}}_1, \hat{\mathcal{X}}_2.\, \mathcal{X}_1 = [\![\hat{\mathcal{X}}_1]\!] \wedge \mathcal{X}_2 = [\![\hat{\mathcal{X}}_2]\!] \wedge \hat{\mathcal{X}}_1 \sqsubseteq \hat{\mathcal{X}}_2$$

We parametrise the consistency models in our semantics. A *consistency model* contains two parts, a *resolution policy* and a *consistency guarantee* (Def. 0.2) [**?**]. Given a abstract execution $\mathcal{X}$, a set of observable transactions T and an address a, the *resolution policy* $\mathcal{P}(\mathcal{X}, T, a)$ decides the observable values for the address a through some computation on the observable transactions T. A common resolution policy is *last-write-win* that if a transaction observes several writes for the same address, it always reads the last write (by the arbitration order). The consistency guarantee gives the minimum constraint for the visibility relation.

Definition 0.2 (Consistency Models)**.** A *Consistency model* is a tuple, $\mathcal{M} = (\mathcal{P}, \mathcal{F}) \in \textsc{ConsisModels}$, including a *resolution policy* and a *consistency guarantee*. A *resolution policy* $\mathcal{P}$ is a function such that for a given address a and a set of observable transactions from a abstract execution, it returns a set of possible observable values.

$$\textsc{ResPos} \triangleq \left\{\mathcal{P} \;\middle|\; \begin{array}{l} \mathcal{P} \in \textsc{AbsExecs} \times \mathcal{P}\left(\textsc{TransID}\right) \times \textsc{Addr} \to \mathcal{P}\left(\textsc{Val}\right) \\ \quad \wedge \forall \mathcal{X}, T, a.\, \mathcal{P}(\mathcal{X}, T, a) \downarrow \implies T \subseteq \mathrm{dom}(\mathcal{X}|_{\mathcal{T}}) \end{array}\right\}$$

A *consistency guarantee* $\mathcal{F}$ is a function such that for an abstract execution, it returns a relation which corresponds to the minimum visibility relation.

$$\textsc{ConGuar} \triangleq \left\{\mathcal{F} \;\middle|\; \begin{array}{l} \mathcal{F} \in \textsc{AbsExecs} \to \mathcal{P}\left(\textsc{TransID} \times \textsc{TransID}\right) \wedge \forall \mathcal{X}. \\ \quad \mathcal{F}(\mathcal{X}) \subseteq \mathcal{X}|_{\mathtt{ar}} \wedge \forall t, t'.\, (t, t') \in \mathcal{F}(\mathcal{X}) \wedge t, t' \in \mathrm{dom}(\mathcal{X}|_{\mathcal{T}}) \end{array}\right\}$$

Note that a well-formed consistency guarantee must not violate the arbitrary order. The order $\sqsubseteq$ between two consistency model $\mathcal{M}_1, \mathcal{M}_2$ is defined as the follows,

$$\begin{array}{rcl} (\mathcal{P}_1, \mathcal{F}_1) \sqsubseteq (\mathcal{P}_2, \mathcal{F}_2) & \overset{\text{def}}{\iff} & \forall \mathcal{X}, T, a. \\ & & \quad \mathcal{P}_2(\mathcal{X}, T, a) \subseteq \mathcal{P}_1(\hat{\mathcal{X}}, T, a) \wedge \mathcal{F}_1(\mathcal{X}) \subseteq \mathcal{F}_2(\mathcal{X}) \end{array}$$

The bottom element is $\bot \triangleq (\lambda(\mathcal{X}, T, a).\, \textsc{Val}, \lambda(\mathcal{X}).\, \emptyset)$, which means one is able to observe any arbitrary value for each address and there is no constraint for visibility relation.

Definition 0.3 (Thread transition labels)**.** Given the set of thread identifiers ThreadID (Def. 0.1), the set of *thread transition labels*, $\iota \in \textsc{Label}$, is defined by the following grammar, where $\mathtt{P}$ denotes a program (Def. **??**), the t demotes a transaction identifier and σ denotes a thread stack (Def. **??**),

$$\iota \in \textsc{Label} \quad ::= \quad \mathtt{id} \mid \mathtt{cmt}(t) \mid \mathtt{fork}(i, \mathtt{P}) \mid \mathtt{join}(i, \sigma)$$

Definition 0.4 (Thread semantics)**.** Given the thread identifiers $i \in \textsc{ThreadID}$, the set of *intermediate programs*, $\hat{\mathtt{P}} \in \textsc{IProg}$, is defined by the following grammar:

$$\hat{\mathtt{P}} ::= \mathtt{P} \mid \hat{\mathtt{P}}\,;\mathtt{wait}(i)$$

Given the set of consistency model ConsisModels (Def. 0.2), the set of thread stacks ThdStks (Def. **??**) and runtime abstract executions RunAbsExecs (Def. 0.1), the *per-thread operational semantics* of programs,

$$\begin{array}{rcl} \to & : & \textsc{ConsisModels} \times \textsc{ThreadID} \times \\ & & \quad ((\textsc{ThdStks} \times \textsc{RunAbsExecs}) \times \textsc{IProg}) \times \textsc{Label} \times ((\textsc{ThdStks} \times \textsc{RunAbsExecs}) \times \textsc{IProg}) \end{array}$$

is defined in Fig. 2.

The PCommit rule "substitutes" the dummy node for the thread i in the runtime abstract execution $\hat{\mathcal{X}}$ with a newly allocated transaction t with its associated events $\mathcal{O}$. To obtain the events $\mathcal{O}$, it prophesies a set of observable transactions T, which will be added into the visibility relation later. Given the observable transactions, the obs_states function computes possible initial heaps, by applying the resolution policy $\mathcal{P}$ for each address. Note that the resolution policy might return more than one value for an address, so there are more than many possible initial heaps. Given the transaction code $\mathtt{T}$, first pick an initial heap h and then given the transaction semantics (Fig. **??**), we can get the events set $\mathcal{O}$. To *extend the runtime abstract execution*, it replaces the dummy node i with the new transaction t, links all transaction from the observable transactions T to the new transaction, and puts the new transaction at the end of arbitration order. Then, it extends the program order by adding back the dummy node i after the transaction t for preserving program order for the future transactions from the same thread.

The Par rule forks a new thread and inserts the appropriate joining point by appending the auxiliary $\mathtt{wait}(i)$ command, where the parameter i denotes the identifier of the newly forked thread. The Wait rule dually awaits the termination of the child thread i indicated by the auxiliary $\mathtt{wait}(i)$ command, and subsequently updates the thread stack. Note that these two rules are labelled with the $\mathtt{fork}(i, \mathtt{P})$ and $\mathtt{join}(i, \sigma)$ which are used by the semantics of the thread pool described shortly (Fig. 3).

In order to model concurrency, we use thread pools. A *thread pool* is a finite partial function from thread identifiers to triples of the form $(\sigma, \hat{\mathtt{P}})$. That is, each thread is associated with a thread stack σ and an intermediate program $\hat{\mathtt{P}}$ to be executed.

Definition 0.5 (Thread pools)**.** Given the set of thread stacks ThdStks (Def. **??**) and intermediate programs IProg (Def. 0.4), a *thread pool* is a a finite partial function from thread identifiers to triples of thread stacks and intermediate programs, $\eta \in \text{TPool} \triangleq \text{ThreadID} \xrightarrow{fin} \text{ThdStks} \times \text{IProg}$.

Definition 0.6 (Thread pool semantics)**.** Given the set of consistent models ConsisModels (Def. 0.2), runtime abstract executions RunAbsExecs (Def. 0.1), transition labels Label (Def. 0.3) and thread pools TPool (Def. 0.5), the *thread pool semantics*,

$$\Rightarrow : \text{ConsisModels} \times (\text{RunAbsExecs} \times \text{TPool}) \times \text{Label} \times (\text{RunAbsExecs} \times \text{TPool})$$

is defined in Fig. 3.

The thread pool operational semantics is given in Fig. 3, where an arbitrary thread in the pool η is picked to run for one step. If the next execution step is a thread fork, then a new thread i' is allocated in the pool to be executed with its thread stack copied from its parent (forking) thread. Conversely, when the next execution step is the joining of thread i', then i' is removed from the thread pool and the stack from the child thread merges into the parent thread.

0.2 Soundness and Completeness

Given a consistency model, we can define the set of abstract executions that satisfy a consistency model (Def. 0.7) [**? ? ?**]. The Lemma 0.8 is for sanity check.

Definition 0.7 (Valid abstract executions)**.** Given a consistency model $(\mathcal{P}, \mathcal{F}) = \mathcal{M} \in \text{ConsisModels}$ (Def. 0.2), the set of valid abstract executions under the model, denoted by $[\![\mathcal{M}]\!]$, is defined as the follows,

$$[\![(\mathcal{P}, \mathcal{F})]\!] \triangleq \left\{ \mathcal{X} = (\mathcal{T}, \mathtt{po}, \mathtt{vis}, \mathtt{ar}) \;\middle|\; \begin{array}{l} \mathcal{F}(\mathcal{X}) \subseteq \mathtt{vis} \wedge \forall t, a, v. \\ \quad (\mathtt{R}, a, v) \in \mathcal{T}(t) \implies v \in \mathcal{P}(\mathcal{X}, \mathcal{X}|_{\mathtt{vis}}(t), a) \end{array} \right\}$$

where $\mathcal{X}|_{\mathtt{vis}}(t)$ returns all the predecessors of t with respect to the visibility relation. This is, for any relation R,

$$R(x) \triangleq \{x' \mid (x', x) \in R\}$$

Lemma 0.8 (Consistency models monotonicity)**.** The abstract executions allowed by a stronger consistency model is allowed by a weaker consistency model, this is,

$$\mathcal{M}_1 \sqsubseteq \mathcal{M}_2 \implies [\![\mathcal{M}_2]\!] \subseteq [\![\mathcal{M}_1]\!]$$

Proof. For any abstract execution $\mathcal{X}$ that satisfies the stronger consistency model $\mathcal{M}_2 = (\mathcal{P}_2, \mathcal{F}_2)$, first, it has $\mathcal{F}_2(\mathcal{X}) \subseteq \mathcal{X}|_{\mathtt{vis}}$ by the Def. 0.7. Given the hypothesis $\mathcal{M}_1 \sqsubseteq \mathcal{M}_2$ and the order definition (Def. 0.2), we know $\mathcal{F}_1(\mathcal{X}) \subseteq \mathcal{F}_2(\mathcal{X})$ so that $\mathcal{F}_1(\mathcal{X}) \subseteq \mathcal{X}|_{\mathtt{vis}}$. Second, assume a transaction t and any of its read event with an address a and a v such that $(\mathtt{R}, a, v) \in \mathcal{T}(t)$ therefore $v \in \mathcal{P}_2(\mathcal{X}, \mathcal{X}|_{\mathtt{vis}}(t), a)$. Similarly by the hypothesis $\mathcal{M}_1 \sqsubseteq \mathcal{M}_2$, we know $\mathcal{P}_2(\mathcal{X}, \mathcal{X}|_{\mathtt{vis}}(t), a) \subseteq \mathcal{P}_1(\mathcal{X}, \mathcal{X}|_{\mathtt{vis}}(t), a)$, thus $v \in \mathcal{P}_1(\mathcal{X}, \mathcal{X}|_{\mathtt{vis}}(t), a)$. Therefore we have $\mathcal{X} \in [\![\mathcal{M}_1]\!]$. □

$$\frac{t \in \mathsf{fresh}\left(\hat{\mathcal{X}}\right) \quad T \subseteq \mathrm{dom}(\hat{\mathcal{X}}\lfloor_{\mathcal{T}}) \quad h \in \mathsf{obs\_states}\left(\hat{\mathcal{X}}, T, \mathcal{P}\right) \quad \sigma \vdash (\emptyset, h, \emptyset), \mathtt{T} \rightsquigarrow^* (\tau, h', \mathcal{O}), \mathtt{skip} \quad \hat{\mathcal{X}}' = \mathsf{new\_abs\_exec}\left(\hat{\mathcal{X}}, i, \mathcal{O}, t, A, \mathcal{F}\right)}{(\mathcal{P}, \mathcal{F}), i \vdash (\sigma, \hat{\mathcal{X}}), [\mathtt{T}] \ \rightarrow_{\mathtt{cmt}(t)} \ (\sigma[\mathtt{ret} \mapsto \tau(\mathtt{ret})], \hat{\mathcal{X}}'), \mathtt{skip}} \ \textsc{PCommit}$$

$$\frac{v = [\![\mathtt{E}]\!]_\sigma \qquad \mathtt{x} \in \textsc{ThdVars}}{\mathcal{M}, i \vdash (\sigma, \hat{\mathcal{X}}), \mathtt{x} := \mathtt{E} \ \rightarrow_{\mathtt{id}} \ (\sigma[\mathtt{x} \mapsto v], \hat{\mathcal{X}}), \mathtt{skip}} \ \textsc{PAssign}$$

$$\frac{[\![\mathtt{E}]\!]_\sigma = 0}{\mathcal{M}, i \vdash (\sigma, \hat{\mathcal{X}}), \mathtt{assume(E)} \ \rightarrow_{\mathtt{id}} \ (\sigma, \hat{\mathcal{X}}), \mathtt{skip}} \ \textsc{PAssume}$$

$$\frac{\mathtt{P}' \in \{\mathtt{P}_1, \mathtt{P}_2\}}{\mathcal{M}, i \vdash (\sigma, \hat{\mathcal{X}}), \mathtt{P}_1 + \mathtt{P}_2 \ \rightarrow_{\mathtt{id}} \ (\sigma, \hat{\mathcal{X}}), \mathtt{P}'} \ \textsc{PChoice}$$

$$\frac{}{\mathcal{M}, i \vdash (\sigma, \hat{\mathcal{X}}), \mathtt{P}^* \ \rightarrow_{\mathtt{id}} \ (\sigma, \hat{\mathcal{X}}), \mathtt{skip} + (\mathtt{P} \,;\, \mathtt{P}^*)} \ \textsc{PLoop}$$

$$\frac{}{\mathcal{M}, i \vdash (\sigma, \hat{\mathcal{X}}), \mathtt{skip} \,;\, \hat{\mathtt{P}} \ \rightarrow_{\mathtt{id}} \ (\sigma, \hat{\mathcal{X}}), \hat{\mathtt{P}}} \ \textsc{PSeqSkip}$$

$$\frac{\mathcal{M}, i \vdash (\sigma, \hat{\mathcal{X}}), \hat{\mathtt{P}}_1 \ \rightarrow_{\iota} \ (\sigma', \hat{\mathcal{X}}'), \hat{\mathtt{P}}_1'}{\mathcal{M}, i \vdash (\sigma, \hat{\mathcal{X}}), \hat{\mathtt{P}}_1 \,;\, \hat{\mathtt{P}}_2 \ \rightarrow_{\iota} \ (\sigma', \hat{\mathcal{X}}'), \hat{\mathtt{P}}_1' \,;\, \hat{\mathtt{P}}_2} \ \textsc{PSeq}$$

$$\frac{\hat{\mathcal{X}}' = \mathsf{extend\_thread}\left(\hat{\mathcal{X}}, i, i'\right)}{\mathcal{M}, i \vdash (\sigma, \hat{\mathcal{X}}), \mathtt{P}_1 \parallel \mathtt{P}_2 \ \rightarrow_{\mathtt{fork}(i', \mathtt{P}_2)} \ (\sigma, \hat{\mathcal{X}}'), \mathtt{P}_1 \,;\, \mathtt{wait}(i')} \ \textsc{PPar}$$

$$\frac{\sigma = \sigma_1 \uplus \sigma_f \qquad \sigma' = \sigma_2 \uplus \sigma_f \qquad \hat{\mathcal{X}} = \mathsf{erase\_thread}\left(\hat{\mathcal{X}}', i'\right)}{\mathcal{M}, i \vdash (\sigma, \hat{\mathcal{X}}), \mathtt{wait}(i') \ \rightarrow_{\mathtt{join}(i', \sigma')} \ (\sigma_1 \uplus \sigma_2 \uplus \sigma_f, \hat{\mathcal{X}}'), \mathtt{skip}} \ \textsc{PWait}$$

SX: The stack could be polluted by the child thread, but they must agree?

where,

$$\begin{aligned}
\mathtt{consis}\left(\hat{\mathcal{X}}, i, t, \mathcal{F}\right) &\triangleq i \in \hat{\mathcal{X}}\lfloor_{\mathcal{I}} \wedge \forall t'.\, ((t', t) \in \mathcal{F}(\|\hat{\mathcal{X}}\|) \implies (t', t) \in \hat{\mathcal{X}}\lfloor_{\mathtt{vis}}) \\
\mathsf{new_abs_exec} &: \begin{pmatrix} \textsc{RunAbsExecs} \times \textsc{ThreadID} \times \textsc{TransID} \times \\ \mathcal{P}(\textsc{Ops}) \times \mathcal{P}(\textsc{TransID}) \times \textsc{ConGuar} \end{pmatrix} \rightharpoonup \textsc{RunAbsExecs} \\
\mathsf{new_abs_exec}\left(\hat{\mathcal{X}}, i, t, \mathcal{O}, T, \mathcal{F}\right) &\triangleq \begin{cases} \hat{\mathcal{X}}' & \mathtt{consis}\left(\hat{\mathcal{X}}', i, t, \mathcal{F}\right) \\ \text{undefined} & \text{otherwise} \end{cases} \\
\hat{\mathcal{X}}' &\equiv \begin{pmatrix} \hat{\mathcal{X}}\lfloor_{\mathcal{T}} \uplus \{t \mapsto \mathcal{O}\}, \hat{\mathcal{X}}\lfloor_{\mathcal{I}}, \hat{\mathcal{X}}\lfloor_{\mathtt{po}} \uplus \{(t', t) \mid (t', i) \in \hat{\mathcal{X}}\lfloor_{\mathtt{po}}\} \uplus \{(t, i)\}, \\ \hat{\mathcal{X}}\lfloor_{\mathtt{vis}} \uplus \{(t', t) \mid t' \in T\}, \hat{\mathcal{X}}\lfloor_{\mathtt{ar}} \uplus \{(t', t) \mid t' \in \mathrm{dom}(\hat{\mathcal{X}}\lfloor_{\mathcal{T}})\} \end{pmatrix} \\
\mathsf{obs_states} &: \textsc{RunAbsExecs} \times \mathcal{P}(\textsc{TransID}) \times \textsc{ResPos} \rightharpoonup \mathcal{P}(\textsc{FPHeap}) \\
\mathsf{obs_states}\left(\hat{\mathcal{X}}, T, \mathcal{P}\right) &\triangleq \{h \mid \forall a, v.\, v \in \mathcal{P}(\|\hat{\mathcal{X}}\|, T, a) \iff h(a) = v\} \\
\mathsf{fresh}\left(\hat{\mathcal{X}}\right) &\triangleq \{t \mid \neg t \in \mathrm{dom}(\hat{\mathcal{X}}\lfloor_{\mathcal{T}})\} \\
\mathsf{extend_thread}\left(\hat{\mathcal{X}}, i, i'\right) &\triangleq (\hat{\mathcal{X}}\lfloor_{\mathcal{T}}, \hat{\mathcal{X}}\lfloor_{\mathcal{I}} \uplus \{i'\}, \hat{\mathcal{X}}\lfloor_{\mathtt{po}} \uplus \{(t, i') \mid (t, i) \in \hat{\mathcal{X}}\lfloor_{\mathtt{po}}\}, \hat{\mathcal{X}}\lfloor_{\mathtt{vis}}, \hat{\mathcal{X}}\lfloor_{\mathtt{ar}}) \\
\mathsf{erase_thread}\left(\hat{\mathcal{X}}, i\right) &\triangleq (\hat{\mathcal{X}}\lfloor_{\mathcal{T}}, \hat{\mathcal{X}}\lfloor_{\mathcal{I}} \setminus \{i\}, \hat{\mathcal{X}}\lfloor_{\mathtt{po}} \setminus \{(t, i) \mid t \in \mathrm{dom}(\hat{\mathcal{X}}\lfloor_{\mathcal{T}})\}, \hat{\mathcal{X}}\lfloor_{\mathtt{vis}}, \hat{\mathcal{X}}\lfloor_{\mathtt{ar}})
\end{aligned}$$

Figure 2: Per-thread operational semantics

$$\frac{\mathcal{M}, i \vdash (\sigma, \hat{\mathcal{X}}), \hat{\mathtt{P}} \ \rightarrow_{\iota} \ (\sigma', \hat{\mathcal{X}}'), \hat{\mathtt{P}}' \quad \iota \in \{\mathtt{id}, \mathtt{cmt}(-)\}}{\mathcal{M} \vdash (\hat{\mathcal{X}}, \eta \uplus \{i \mapsto (\sigma, \hat{\mathtt{P}})\}) \ \Rightarrow_{\iota} \ (\hat{\mathcal{X}}, \eta \uplus \left\{i \mapsto (\sigma', \hat{\mathtt{P}}')\right\})} \ \textsc{PSingle}$$

$$\frac{\mathcal{M}, i \vdash (\sigma, \hat{\mathcal{X}}), \hat{\mathtt{P}} \ \rightarrow_{\mathtt{fork}(i', \mathtt{P})} \ (\sigma, \hat{\mathcal{X}}'), \hat{\mathtt{P}}'}{\mathcal{M} \vdash (\hat{\mathcal{X}}, \eta \uplus \{i \mapsto (\sigma, \hat{\mathtt{P}})\}) \ \Rightarrow_{\mathtt{fork}(i', \mathtt{P})} \ (\hat{\mathcal{X}}', \eta \uplus \left\{i \mapsto (\sigma, \hat{\mathtt{P}}'), i' \mapsto (\sigma, \mathtt{P})\right\})} \ \textsc{PFork}$$

$$\frac{\mathcal{M}, i \vdash (\sigma, \hat{\mathcal{X}}), \hat{\mathtt{P}} \ \rightarrow_{\mathtt{join}(i', \sigma')} \ (\sigma'', \hat{\mathcal{X}}'), \hat{\mathtt{P}}'}{\mathcal{M} \vdash (\hat{\mathcal{X}}, \eta \uplus \{i \mapsto (\sigma, \hat{\mathtt{P}}), i' \mapsto (\sigma', \mathtt{skip})\}) \ \Rightarrow_{\mathtt{join}(i', \sigma'')} \ (\hat{\mathcal{X}}', \eta \uplus \left\{i \mapsto (\sigma'', \hat{\mathtt{P}}')\right\})} \ \textsc{PJoin}$$

Figure 3: Thread pool semantics

Theorem 0.9 (Soundness of the semantics)**.** For any runtime abstract abstract execution $\hat{\mathcal{X}}$ that satisfies a consistency model $\mathcal{M}$, if the semantics under consistency model $\mathcal{M}$ take one step to a new abstract execution $\hat{\mathcal{X}}'$, the new execution should satisfy the consistency model. This is,

$$\begin{aligned}&\forall \mathcal{M}, i, \hat{\mathcal{X}}, \hat{\mathcal{X}}', \sigma, \sigma', \hat{\mathtt{P}}, \hat{\mathtt{P}}', \iota.\\ &\quad [\![\hat{\mathcal{X}}]\!] \in [\![\mathcal{M}]\!] \wedge \mathcal{M}, i \vdash (\hat{\mathcal{X}}, \sigma), \hat{\mathtt{P}} \rightarrow_{\iota} (\hat{\mathcal{X}}', \sigma'), \hat{\mathtt{P}}' \implies [\![\hat{\mathcal{X}}']\!] \in [\![\mathcal{M}]\!]\end{aligned}$$

Proof. We prove it by induction on the derivations.

Base Case PCommit.

By the PCommit rule, it has $\hat{\mathtt{P}} = [\mathtt{T}]$, $\hat{\mathtt{P}}' = \mathtt{skip}$ and $\iota = \mathtt{cmt}(t)$, for some transaction code $\mathtt{T}$ and identifier t. Let variables $\mathcal{X} = [\![\hat{\mathcal{X}}]\!]$ and $\mathcal{X}' = [\![\hat{\mathcal{X}}']\!]$ in the following discussion. We need to prove the follows,

$$\begin{aligned}&\forall t', t'', a, v.\\ &\quad (\mathtt{R}, a, v) \in \mathcal{X}'\lfloor_{\mathcal{T}}(t') \implies v \in \mathcal{P}(\mathcal{X}', \mathcal{X}'\lfloor_{\mathtt{vis}}(t'), a) \qquad (0.1)\\ &\quad (t', t'') \in \mathcal{F}(\mathcal{X}') \implies (t', t'') \in \mathcal{X}'\lfloor_{\mathtt{vis}} \qquad (0.2)\end{aligned}$$

First for the Eq. (0.1), it only needs to check the new transaction t as others are proved directly from the hypothesis. Given an initial heap h, a set of events $\mathcal{O}$ associated with the new transaction t, a set of transactions T observed by the new transaction t, assume these variables satisfy the follows,

$$\begin{aligned}&\exists \sigma, \tau, h'.\\ &\quad \sigma \vdash (\emptyset, h, \emptyset), \mathtt{T} \rightsquigarrow^{*} (\tau, h', \mathcal{O}), \mathtt{skip} \wedge h \in \mathsf{obs\_states}\left(\hat{\mathcal{X}}, T, \mathcal{P}\right)\end{aligned}$$

therefore the following hold,

$$(\mathtt{R}, a, v) \in \mathcal{O} \implies h(a) = v$$

Because by the transaction semantics (Fig. **??**), a transaction only records the first read event for each address. It can be proved by induction on the derivations for transaction operational semantics, where the only rule that involves read event is TRead. If the new read event is included in the events set after flushing read (Def. **??**), *i.e.* $(\mathtt{R}, a, v) \in (\mathcal{O} \lhd (\mathtt{R}, a, v))$, this means there is no other read and write to the same address before, so that the value v associate with the address a is the initial value, *i.e.* $h(a)$, and after that no other read event can over-write. By the obs_states function (Fig. 2), where $h(a) = v \iff \mathcal{P}(\mathcal{X}, T, a)$, the following hold,

$$(\mathtt{R}, a, v) \in \mathcal{O} \implies \mathcal{P}(\mathcal{X}, T, a)$$

Since the new_abs_exec function only extends abstract execution, which means $\mathcal{X} \sqsubseteq \mathcal{X}'$, so that,

$$(\mathtt{R}, a, v) \in \mathcal{O} \implies \mathcal{P}(\mathcal{X}', T, a)$$

By the obs_states function, we have $\mathcal{O} = \mathcal{X}'\lfloor_{\mathcal{T}}(t)$ and $T = \mathcal{X}'\lfloor_{\mathtt{vis}}(t)$ therefore we prove Eq. (0.1).

Second for Eq. (0.2), the visibility relation of the new abstract execution $\mathcal{X}'\lfloor_{\mathtt{vis}}$ contains the minimum relation required by the consistency guarantee. Similarly, it is sufficient to consider those visibility edges related to the new transaction t. Note that for any transaction $t'' \in \mathrm{dom}(\mathcal{X}'\lfloor_{\mathcal{T}})$, it has $(t, t'') \notin \mathcal{F}(\mathcal{X}')$. Given new_abs_exec function, so that $(t'', t) \in \mathcal{X}'\lfloor_{\mathtt{ar}}$. Then, given the consistency guarantee (Def. 0.2), it cannot violate arbitration order, this is, $\mathcal{F}(\mathcal{X}') \subseteq \mathcal{X}'\lfloor_{\mathtt{ar}}$. Thus, it is safe to assume a transaction $t' \in \mathrm{dom}(\mathcal{X}'\lfloor_{\mathcal{T}})$ such that $(t', t) \in \mathcal{F}(\mathcal{X}')$. By the `consis` predicate, we have $(t', t) \in \mathcal{F}(\mathcal{X}') \implies (t', t) \in \mathcal{X}'\lfloor_{\mathtt{vis}}$, so that we prove Eq. (0.2).

Base Case PAssign, PAssume, PChoice, PLoop, PSeqSkipS.

For these base cases, the runtime abstract execution remains the same, *i.e.* $\hat{\mathcal{X}} = \hat{\mathcal{X}}'$, so they trivially hold because of the hypothesis.

Base Case PPar, PWait.

For these two base cases, since the extend_thread and erase_thread functions only change relations related to the corresponding threads, therefore *i.e.* $[\![\hat{\mathcal{X}}]\!] = [\![\hat{\mathcal{X}}']\!]$, so they hold because of the hypothesis.

Inductive Case PSeq.

It is proved directly by applying the I.H. □

For sanity check and also proving the completeness of this semantics, we first prove the semantics is monotonic (Lemma 0.10), which means that for any reduction that can happen in the stronger consistency model, it can also happen in the weaker one.

Lemma 0.10 (Semantics monotonicity)**.** Given an initial runtime abstract execution $\hat{\mathcal{X}}$, if it can transfer to an abstract execution $\hat{\mathcal{X}}'$ by reducing one step of the semantics under stronger consistency model $\mathcal{M}_2$, it is also possible by reducing one step of the semantics under a weaker consistency model $\mathcal{M}_1$.

$$\begin{aligned}&\forall \mathcal{M}_1, \mathcal{M}_2, i, \hat{\mathcal{X}}, \hat{\mathcal{X}}', \hat{\mathtt{P}}, \hat{\mathtt{P}}', \sigma, \sigma', \iota.\\ &\quad \mathcal{M}_2, i \vdash (\hat{\mathcal{X}}, \sigma), \hat{\mathtt{P}} \to_\iota (\hat{\mathcal{X}}', \sigma'), \hat{\mathtt{P}}' \wedge \mathcal{M}_1 \sqsubseteq \mathcal{M}_2\\ &\quad \implies \mathcal{M}_1, i \vdash (\hat{\mathcal{X}}, \sigma), \hat{\mathtt{P}} \to_\iota (\hat{\mathcal{X}}', \sigma'), \hat{\mathtt{P}}'\end{aligned}$$

Proof. We prove it by induction on the derivations. The only interesting case is the PCommit rule.

Base Case PCommit.

Let variables $(\mathcal{P}_1, \mathcal{F}_1) = \mathcal{M}_1$ and $(\mathcal{P}_2, \mathcal{F}_2) = \mathcal{M}_2$ respectively. Given an initial runtime abstract execution $\hat{\mathcal{X}}$, a set of observable transactions T, a new transaction identifier t and a thread identifier i, by the PCommit rule (Fig. 3), it is sufficient to prove, first, all the observable states under the stronger consistency model can also be observed under the weaker one,

$$\mathsf{obs\_states}\left(\hat{\mathcal{X}}, T, \mathcal{P}_2\right) \subseteq \mathsf{obs\_states}\left(\hat{\mathcal{X}}, T, \mathcal{P}_1\right) \tag{0.3}$$

and second if the new runtime abstract execution $\hat{\mathcal{X}}'$ exists under the stronger concurrency model, it should also exist under weaker one,

$$\mathtt{consis}\left(\hat{\mathcal{X}}', t, i, \mathcal{F}_2\right) \implies \mathtt{consis}\left(\hat{\mathcal{X}}', t, i, \mathcal{F}_1\right) \tag{0.4}$$

To prove Eq. (0.3), assume an observable heap h under stronger consistency model $\mathcal{M}_2$, which means $h \in \mathsf{obs\_states}\left(\hat{\mathcal{X}}, T, \mathcal{P}_2\right)$. Then, assume an address a and the corresponding value v such that $h(a) = v$. By the obs_states function (Fig. 2) and resolution policy (Def. 0.2), it is known that $v \in \mathcal{P}_2([\![\hat{\mathcal{X}}]\!], T, a)$. Because of $\mathcal{P}_2([\![\hat{\mathcal{X}}]\!], T, a) \subseteq \mathcal{P}_1([\![\hat{\mathcal{X}}]\!], T, a)$ (Def. 0.2), we have $v \in \mathcal{P}_1([\![\hat{\mathcal{X}}]\!], T, a)$, so $h \in \mathsf{obs\_states}\left(\hat{\mathcal{X}}, T, \mathcal{P}_1\right)$. For Eq. (0.4), assume the $\mathtt{consis}\left(\hat{\mathcal{X}}, t, i, \mathcal{F}_2\right)$ predicate holds and assume an edge $(t', t) \in \mathcal{F}_2([\![\hat{\mathcal{X}}']\!])$ for some t'. Thus, the edge (t', t) will be included in the visibility relation of the new runtime abstract execution, *i.e.* $(t', t) \in \hat{\mathcal{X}}'|_{\mathtt{vis}}$. Since $\mathcal{F}_1([\![\hat{\mathcal{X}}]\!]) \subseteq \mathcal{F}_2([\![\hat{\mathcal{X}}]\!])$, so $(t', t) \in \mathcal{F}_1([\![\hat{\mathcal{X}}']\!]) \implies (t', t) \in \hat{\mathcal{X}}'|_{\mathtt{vis}}$ holds. This means for any edges that satisfy the consistency guarantee for stronger model, they also satisfy the weaker consistency guarantee, thus the Eq. (0.4) holds. Combining Eq. (0.3) and Eq. (0.4), we prove PCommit.

Base Case PAssign, PAssume, PChoice, PLoop, PSeqSkipS, PPar, PWait.

These base cases do not depend on the consistency model, so they trivial hold because of the hypothesis.

Inductive Case PSeq.

It is proved directly by applying the I.H. □

Lemma 0.11 (Preservation of the consistency model)**.**

$$\begin{aligned}&\forall \mathcal{M}_1, \mathcal{M}_2, i, \hat{\mathcal{X}}, \hat{\mathcal{X}}', \sigma, \sigma', \hat{\mathtt{P}}, \hat{\mathtt{P}}', \iota.\\ &\quad \mathcal{M}_1, i \vdash (\hat{\mathcal{X}}, \sigma), \hat{\mathtt{P}} \to_\iota (\hat{\mathcal{X}}', \sigma'), \hat{\mathtt{P}}' \wedge [\![\hat{\mathcal{X}}']\!] \in [\![\mathcal{M}_2]\!] \implies [\![\hat{\mathcal{X}}]\!] \in [\![\mathcal{M}_2]\!]\end{aligned}$$

Proof. We prove it by induction on the derivations. The only interesting case is the Commit.

Base Case Commit.

By the rule it has $\hat{\mathtt{P}} = [\mathtt{T}]$, $\hat{\mathtt{P}}' = \mathtt{skip}$ and $\iota = \mathtt{cmt}(t)$. We prove this case by deriving contradiction. Assume $[\![\hat{\mathcal{X}}]\!] \notin [\![\mathcal{M}_2]\!]$, which means that there exists an edge (t', t'') such that it is in the consistency guarantee, $(t', t'') \in \mathcal{F}_2([\![\hat{\mathcal{X}}]\!])$ but not is not included in the visibility relation.

$$(t', t'') \notin \hat{\mathcal{X}}|_{\mathtt{vis}} \tag{0.5}$$

Another possibility is that there is a read event from a transaction, $(\mathtt{W}, a, v) \in \hat{\mathcal{X}}\lfloor_{\mathcal{T}}\ (t')$ where the value is not observable under the stronger consistency model, *i.e.*

$$v \notin \mathcal{P}_2(\llbracket \hat{\mathcal{X}} \rrbracket, \hat{\mathcal{X}}\lfloor_{\mathtt{vis}}\ (t'), a) \tag{0.6}$$

Because the rule only extend the runtime abstract execution $\hat{\mathcal{X}} \sqsubseteq \hat{\mathcal{X}}'$, this means the edge (t', t'') is not in the runtime abstract execution after reduction $\hat{\mathcal{X}}'$, or the transaction t' reads a unobservable value,

$$(t', t'') \notin \hat{\mathcal{X}}'\lfloor_{\mathtt{vis}} \vee v \notin \mathcal{P}_2(\llbracket \hat{\mathcal{X}}' \rrbracket, \hat{\mathcal{X}}'\lfloor_{\mathtt{vis}}\ (t'), a)$$

Both cases lead to $\hat{\mathcal{X}}' \notin [\![\mathcal{M}_2]\!]$ so there is contradiction to the hypothesis. Therefore we have the proof for this base case.

Base Case PAssign, PAssume, PChoice, PLoop, PSeqSkipS.

For these base cases, the runtime abstract execution remains the same, *i.e.* $\hat{\mathcal{X}} = \hat{\mathcal{X}}'$, so they trivially hold because of the hypothesis.

Base Case PPar, PWait.

For these two base cases, since the extend_thread and erase_thread functions change edges only related to the corresponding threads, therefore *i.e.* $\llbracket \hat{\mathcal{X}} \rrbracket = \llbracket \hat{\mathcal{X}}' \rrbracket$, so it holds because of the hypothesis.

Inductive Case PSeq.

It is proved directly by applying the I.H. □

The completeness means that if an abstract execution $\hat{\mathcal{X}}$ satisfies a consistency model, it always is possible to produce such execution through the semantics under the corresponding consistency model for some initial configurations. To define the completeness, we introduce *anarchic semantics*, which is the semantics under the bottom element for consistency model $\bot$ as for the fact that there is no constraint for the visibility relations and for the observable value for each address.

SX: citation for anarchic semantics?

The Theorem 0.12 says, given some initial configurations, after one step under the anarchic semantics, if one is "lucky" that it ends up with a runtime abstract execution $\hat{\mathcal{X}}'$ that satisfies a consistency model $\mathcal{M}$, it is possible to get the same result using the semantics specifically for the consistency model $\mathcal{M}$.

Theorem 0.12 (Completeness of the semantics)**.** For any initial configuration $((\hat{\mathcal{X}}, \sigma), \hat{\mathtt{P}})$, after one step under the *anarchic semantics*, *i.e.* the semantics under the bottom element $\bot$, it ends up with $\hat{\mathcal{X}}'$, and if the new runtime abstract execution $\hat{\mathcal{X}}'$ satisfies a consistency model $\mathcal{M}$, then there is a corresponding step using semantics under consistency model $\mathcal{M}$.

$$\begin{aligned}&\forall \mathcal{M}, i, \hat{\mathcal{X}}, \hat{\mathcal{X}}', \sigma, \sigma', \hat{\mathtt{P}}, \hat{\mathtt{P}}', \iota.\\ &\quad \bot, i \vdash (\hat{\mathcal{X}}, \sigma), \hat{\mathtt{P}} \to_\iota (\hat{\mathcal{X}}', \sigma'), \hat{\mathtt{P}}' \wedge \llbracket \hat{\mathcal{X}}' \rrbracket \in [\![\mathcal{M}]\!]\\ &\quad\quad \Longrightarrow \mathcal{M}, i \vdash (\hat{\mathcal{X}}, \sigma), \hat{\mathtt{P}} \to_\iota (\hat{\mathcal{X}}', \sigma'), \hat{\mathtt{P}}'\end{aligned}$$

Proof. For any runtime abstract execution after one step such that $\llbracket \hat{\mathcal{X}}' \rrbracket \in [\![\mathcal{M}]\!]$, by the Lemma 0.11, it is known that the initial configuration also satisfies the consistency model, this is, $\llbracket \hat{\mathcal{X}} \rrbracket \in [\![\mathcal{M}]\!]$. Since $\bot$ is the bottom element such that $\bot \sqsubseteq \mathcal{M}$, because the semantics are monotonic with respect to the order of consistency model (Lemma 0.10), we have the proof. □