1 semantics

We model the state of a database by a time-stamp heap that is a partial function from locations to their histories. The history is a partial function from times to a set of events. A event is a triple consisting of the value being read or written, the operation, i.e. either read or write, and the transaction identifier. We use thread pool to model the concurrency. Each thread has a local stack and a local time, but a globally shared time-stamp heap. Therefore a thread pool is a partial functions from thread identifiers to the corresponding stack, time and transactions. The state of each transaction, i.e. local state, are a stack which is shared between transactions from the same thread, a heap that is a snapshot of the time-stamp heap, and fingerprints that are the heap locations being read and written.

$$
\begin{array}{rcl}
l \in \mathtt{Loc} & \triangleq & \mathbb{N} \\
v \in \mathtt{Val} & \triangleq & \mathbb{N} \uplus \mathtt{Loc} \\
\mathtt{Var} & \triangleq & \{\mathtt{x}, \mathtt{y}, \ldots\} \\
t \in \mathtt{TimeStamp} & \triangleq & \mathbb{N} \\
h \in \mathtt{Heap} & \triangleq & \mathtt{Loc} \rightharpoonup \mathtt{Val} \\
s \in \mathtt{Stack} & \triangleq & \mathtt{Var} \to \mathtt{Val} \\
o \in \mathtt{Operation} & \triangleq & \{\mathtt{r}, \mathtt{w}\} \\
\mathcal{T} \subseteq \mathtt{TransID} & \triangleq & \{\alpha, \beta, \ldots\} \\
\mathtt{ThreadID} & \triangleq & \{i, j, \ldots\} \\
rs \in \mathtt{ReadSet}, ws \in \mathtt{WriteSet} & \triangleq & \mathcal{P}(\mathtt{Loc}) \\
\hbar \in \mathtt{TimeStampHeap} & \triangleq & \mathtt{Loc} \rightharpoonup (\mathtt{TimeStamp} \rightharpoonup \mathcal{P}(\mathtt{Val} \times \mathtt{Operation} \times \mathtt{TransID})) \\
(s, \hbar, t) \in \mathtt{ThreadState} & \triangleq & \mathtt{Stack} \times \mathtt{TimeStampHeap} \times \mathtt{TimeStamp} \\
\eta \in \mathtt{ThreadPool} & \triangleq & \mathtt{ThreadID} \rightharpoonup \mathtt{Stack} \times \mathtt{TimeStamp} \times \mathbb{P} \\
\Sigma \in \mathtt{State} & \triangleq & \mathtt{TimeStampHeap} \times \mathtt{ThreadPool} \\
\sigma = (s, h, rs, ws) \in \mathtt{LocalState} & \triangleq & \mathtt{Stack} \times \mathtt{Heap} \times \mathtt{ReadSet} \times \mathtt{WriteSet}
\end{array}
$$

The arithmetic expression and boolean expression are standard and have no side effect.

$$
\mathbb{E} ::= \quad v \quad | \quad \mathtt{x} \quad | \quad \mathbb{E} + \mathbb{E} \quad | \quad \mathbb{E} * \mathbb{E} \quad | \quad \ldots
$$

$$
\begin{array}{rcl}
[\![v]\!]_s & \triangleq & v \\
[\![\mathtt{x}]\!]_s & \triangleq & s(v) \\
[\![\mathbb{E}_1 + \mathbb{E}_2]\!]_s & \triangleq & [\![\mathbb{E}_1]\!]_s + [\![\mathbb{E}_2]\!]_s \\
[\![\mathbb{E}_1 * \mathbb{E}_2]\!]_s & \triangleq & [\![\mathbb{E}_1]\!]_s * [\![\mathbb{E}_2]\!]_s
\end{array}
$$

$$
\mathbb{B} ::= \quad \mathtt{true} \quad | \quad \mathtt{false} \quad | \quad \mathbb{E} = \mathbb{E} \quad | \quad \mathbb{E} < \mathbb{E} \quad | \quad \mathtt{not}\ \mathbb{B} \quad | \quad \mathbb{B}\ \mathtt{and}\ \mathbb{B} \quad | \quad \mathbb{B}\ \mathtt{or}\ \mathbb{B} \quad | \quad \ldots
$$

$$
\begin{array}{rcl}
[\![\mathtt{true}]\!]_s & \triangleq & \mathtt{true} \\
[\![\mathtt{false}]\!]_s & \triangleq & \mathtt{false} \\
[\![\mathbb{E}_1 = \mathbb{E}_2]\!]_s & \triangleq & [\![\mathbb{E}_1]\!]_s = [\![\mathbb{E}_2]\!]_s \\
[\![\mathbb{E}_1 < \mathbb{E}_2]\!]_s & \triangleq & [\![\mathbb{E}_1]\!]_s < [\![\mathbb{E}_2]\!]_s \\
[\![\mathtt{not}\ \mathbb{B}]\!]_s & \triangleq & \neg\, [\![\mathbb{B}]\!]_s \\
[\![\mathbb{B}_1\ \mathtt{and}\ \mathbb{B}_2]\!]_s & \triangleq & [\![\mathbb{B}_1]\!]_s \wedge [\![\mathbb{B}_2]\!]_s \\
[\![\mathbb{B}_1\ \mathtt{or}\ \mathbb{B}_2]\!]_s & \triangleq & [\![\mathbb{B}_1]\!]_s \vee [\![\mathbb{B}_2]\!]_s
\end{array}
$$

$$
\mathbb{C} ::= \begin{array}{l} \mathtt{skip} \quad | \quad \mathtt{x} := \mathbb{E} \quad | \quad [\mathbb{E}] := \mathbb{E} \quad | \quad \mathtt{x} := [\mathbb{E}] \quad | \quad \mathtt{if}\,(\mathbb{B})\ \mathbb{C}\ \mathtt{else}\ \mathbb{C} \quad | \\ \mathtt{while}\,(\mathbb{B})\ \mathbb{C} \quad | \quad \mathbb{C}\,;\mathbb{C} \end{array}
$$

The syntax and semantics of a single transaction are standard except *mutate* and *deref*. The *mutate* also adds the location being written to the write fingerprint set ws and the *deref* adds the location to rs. Note that there is no parallel composition, because it is within a transaction.

$$
(-,-) \ \rightsquigarrow_l \ (-,-) \quad \triangleq \quad (\mathtt{LocalState} \times \mathbb{C}) \times (\mathtt{LocalState} \times \mathbb{C})
$$

$$
\frac{[\![\mathbb{E}]\!]_s = v}{(s, h, rs, ws), \mathtt{x} := \mathbb{E} \ \rightsquigarrow_l \ (s[\mathtt{x} \mapsto v], h, rs, ws), \mathtt{skip}}\ ass
$$

$$\frac{[\![\mathbb{E}_1]\!]_s = l \quad [\![\mathbb{E}_2]\!]_s = v \quad l \in \mathrm{dom}(h)}{(s,h,rs,ws), [\mathbb{E}_1] := \mathbb{E}_2 \ \rightsquigarrow_l \ (s, h[l \mapsto v], rs, ws \cup \{l\}), \mathtt{skip}} \ mutate$$

$$\frac{[\![\mathbb{E}]\!]_s = l \quad v = h(l) \quad l \in \mathrm{dom}(h)}{(s,h,rs,ws), \mathtt{x} := [\mathbb{E}] \ \rightsquigarrow_l \ (s[\mathtt{x} \mapsto v], h, rs \cup \{l\}, ws), \mathtt{skip}} \ deref$$

$$\frac{[\![\mathbb{B}]\!]_s = \mathtt{true}}{(s,h,rs,ws), \mathtt{if}\ (\mathbb{B})\ \mathbb{C}_1\ \mathtt{else}\ \mathbb{C}_2 \ \rightsquigarrow_l \ (s,h,rs,ws), \mathbb{C}_1} \ ifelsetrue$$

$$\frac{[\![\mathbb{B}]\!]_s = \mathtt{false}}{(s,h,rs,ws), \mathtt{if}\ (\mathbb{B})\ \mathbb{C}_1\ \mathtt{else}\ \mathbb{C}_2 \ \rightsquigarrow_l \ (s,h,rs,ws), \mathbb{C}_2} \ ifelsefalse$$

$$\frac{[\![\mathbb{B}]\!]_s = \mathtt{true}}{(s,h,rs,ws), \mathtt{while}\ (\mathbb{B})\ \mathbb{C} \ \rightsquigarrow_l \ (s,h,rs,ws), \mathbb{C}\,;\mathtt{while}\ (\mathbb{B})\ \mathbb{C}} \ whiletrue$$

$$\frac{[\![\mathbb{B}]\!]_s = \mathtt{false}}{(s,h,rs,ws), \mathtt{while}\ (\mathbb{B})\ \mathbb{C} \ \rightsquigarrow_l \ (s,h,rs,ws), \mathtt{skip}} \ whilefalse$$

$$\frac{}{(s,h,rs,ws), \mathtt{skip}\,;\mathbb{C}_2 \ \rightsquigarrow_l \ (s,h,rs,ws), \mathbb{C}_2} \ seqskip$$

$$\frac{(s,h,rs,ws), \mathbb{C}_1 \ \rightsquigarrow_l \ (s',h',rs',ws'), \mathbb{C}_1'}{(s,h,rs,ws), \mathbb{C}_1\,;\mathbb{C}_2 \ \rightsquigarrow_l \ (s',h',rs',ws'), \mathbb{C}_1'\,;\mathbb{C}_2} \ seqnonskip$$

A program is sequential and parallel composition of transactions. To give semantics for a program, we extend the syntax by adding an extra waiting command, $\mathtt{wait}(i)$, as suffix. Intuitively, this $\mathtt{wait}(i)$ indicates that current thread is waiting another thread identified by i until it commits all its transactions and then join the thread.

$$\mathbb{P} ::= \ \mathtt{skip} \ \mid \ [\mathbb{C}] \ \mid \ \mathbb{P}\,;\mathbb{P} \ \mid \ \mathtt{if}\ (\mathbb{B})\ \mathbb{P}\ \mathtt{else}\ \mathbb{P} \ \mid \ \mathtt{while}\ (\mathbb{B})\ \mathbb{P} \ \mid \ \mathbb{P} \parallel \mathbb{P}$$

$$\mathbb{P}^{\uparrow} ::= \ \mathtt{wait}(i) \ \mid \ \mathbb{P} \ \mid \ \mathbb{P}^{\uparrow}\,;\mathtt{wait}(i) \ \mid$$

We will explain the *commit*, *par* and *wait*, and the rest are straightforward. We give label to each transition, and these labels are only usefully for parallel composition.

The *commit* rule says that a transaction prophesies a starting time when this transaction takes a snapshot h_s and runs locally, and an ending time when it successfully commits ensured by the `allowcommit`.

The *par* rule forks a new thread and appends a $\mathtt{wait}(i)$, parametrised by the new thread identifier i, at the merging point. The *wait* rule waits the thread i until it finishes, then joins the thread and updates the time to the maximum between the two threads. Note that these two rules are labelled with $\mathtt{fork}(i, \mathbb{P})$ or $\mathtt{join}(i,t)$ which are used by the semantics of a top level theadpool.

$$\begin{array}{rcl}
\iota \in \mathtt{Label} & \triangleq & \mathtt{id} \ \mid \ \mathtt{cmt}(\alpha) \ \mid \ \mathtt{fork}(i,\mathbb{P}) \ \mid \ \mathtt{join}(i,t) \\
(-,-) \ \bar{\rightsquigarrow}_t \ (-,-) & \triangleq & (\mathtt{ThreadState} \times \mathbb{P}^{\uparrow}) \times \mathtt{Label} \times (\mathtt{ThreadState} \times \mathbb{P}^{\uparrow})
\end{array}$$

$$\begin{array}{rcl}
startstate(\hbar, t) & \triangleq & \lambda l.\, v \\
 & & \mathtt{where}\ \exists t' \le t.\, \hbar(l)(t') = \{(v, \mathtt{w}, -)\} \wedge \forall t'' \in (t', t).\, (-, \mathtt{w}, -) \notin \hbar(l)(t'') \\
\mathtt{allowcommit}(\hbar, ws, rs, t_s, t_e) & \triangleq & \mathtt{atomicop}(\hbar, ws, rs, t_s, t_e) \wedge \mathtt{consistent}(\hbar, ws, rs, t_s, t_e) \\
\mathtt{atomicop}(\hbar, ws, rs, t_s, t_e) & \triangleq & \forall l_w \in ws.\, \hbar(l_w)(t_e) \uparrow \wedge \forall l_r \in rs.\, (-, \mathtt{w}, -) \notin \hbar(l_r)(t_s) \\
\mathtt{consistent}(\hbar, ws, rs, t_s, t_e) & \triangleq & \forall l_w \in ws, t \in (t_s, t_e).\, (-, \mathtt{w}, -) \notin \hbar(l_w)(t) \wedge \\
 & & \forall \alpha.\, \nexists t_{\alpha s} < t_e, t_{\alpha e} > t_e.\, \hbar(l_w)(t_{\alpha s}) = (-, \mathtt{r}, \alpha) \wedge \hbar(l_w)(t_{\alpha e}) = (-, \mathtt{w}, \alpha) \wedge \\
 & & \exists t_{min} = \min(\{t'' \mid t'' > t_e \wedge \hbar(l_w)(t'') \downarrow\}).\, (-, \mathtt{r}, -) \notin \hbar(l_w)(t_{min}) \\
commit(\hbar, h, ws, rs, \alpha, t_s, t_e) & \triangleq & \lambda l. \begin{cases} \hbar(l) & l \notin ws \cup rs \\ \hbar(l)[t_e \mapsto \hbar(l)(t_e) \uplus \{(h(l), \mathtt{w}, \alpha)\}] & l \in ws \\ \hbar(l)[t_s \mapsto \hbar(l)(t_s) \uplus \{(h(l), \mathtt{r}, \alpha)\}] & l \in rs \end{cases} \\
freshTransId(\hbar) & \triangleq & \alpha\ \mathtt{where}\ \alpha \notin \left\{ \alpha' \;\middle|\; (-,-,\alpha') \in \bigcup_{l,t} \hbar(l)(t) \right\}
\end{array}$$

$$\frac{t_s \geq t \quad t_e > t_s \quad h_s = startstate(\hbar, t_s) \quad (s, h_s, \emptyset, \emptyset), \mathbb{C} \rightsquigarrow_l^* (s', h_e, rs, ws), \texttt{skip} \quad \texttt{allowcommit}(\hbar, ws, rs, t_s, t_e) \quad \alpha = freshTransId(\hbar) \quad \hbar' = commit(\hbar, h_e, ws, rs, \alpha, t_s, t_e)}{(s, \hbar, t), [\mathbb{C}] \overset{\texttt{cmt}(\alpha)}{\rightsquigarrow_t} (s', \hbar', t_e), \texttt{skip}} \; commit$$

$$\frac{[\![\mathbb{B}]\!]_s = \texttt{true}}{(s, \hbar, t), \texttt{if}\,(\mathbb{B})\ \mathbb{P}_1\ \texttt{else}\ \mathbb{P}_2 \overset{\texttt{id}}{\rightsquigarrow_t} (s, \hbar, t), \mathbb{P}_1} \; conditiontrue$$

$$\frac{[\![\mathbb{B}]\!]_s = \texttt{false}}{(s, \hbar, t), \texttt{if}\,(\mathbb{B})\ \mathbb{P}_1\ \texttt{else}\ \mathbb{P}_2 \overset{\texttt{id}}{\rightsquigarrow_t} (s, \hbar, t), \mathbb{P}_2} \; conditionfalse$$

$$\frac{[\![\mathbb{B}]\!]_s = \texttt{false}}{(s, \hbar, t), \texttt{while}\,(\mathbb{B})\ \mathbb{P} \overset{\texttt{id}}{\rightsquigarrow_t} (s, \hbar, t), \texttt{skip}} \; norep$$

$$\frac{[\![\mathbb{B}]\!]_s = \texttt{true}}{(s, \hbar, t), \texttt{while}\,(\mathbb{B})\ \mathbb{P} \overset{\texttt{id}}{\rightsquigarrow_t} (s, \hbar, t), \mathbb{P}\,;\texttt{while}\,(\mathbb{B})\ \mathbb{P}} \; rep$$

$$\frac{}{(s, \hbar, t), \texttt{skip}\,;\mathbb{P}^\uparrow \overset{\texttt{id}}{\rightsquigarrow_t} (s, \hbar, t), \mathbb{P}^\uparrow} \; seqskip$$

$$\frac{(s, \hbar, t), \mathbb{P}_1^\uparrow \overset{\iota}{\rightsquigarrow_t} (s', \hbar', t'), \mathbb{P}_1^{\uparrow\prime}}{(s, \hbar, t), \mathbb{P}_1^\uparrow\,;\mathbb{P}_2^\uparrow \overset{\iota}{\rightsquigarrow_t} (s', \hbar', t'), \mathbb{P}_1^{\uparrow\prime}\,;\mathbb{P}_2^\uparrow} \; seqnoskip$$

$$\frac{}{(s, \hbar, t), \mathbb{P}_1 \parallel \mathbb{P}_2 \overset{\texttt{fork}(i, \mathbb{P}_2)}{\rightsquigarrow_t} (s, \hbar, t), \mathbb{P}_1\,;\texttt{wait}(i)} \; par$$

$$\frac{}{(s, \hbar, t), \texttt{wait}(i) \overset{\texttt{join}(i, t')}{\rightsquigarrow_t} (s, \hbar, \max\{t, t'\}), \texttt{skip}} \; wait$$

$$- \overset{-}{\rightsquigarrow_g} - \triangleq \texttt{State} \times \texttt{Label} \times \texttt{State}$$

The semantics of theadpool picks a thread to runs one step. If the step is a fork, it generates a new thread with a new stack and a local time that is the same as its parent thread. If it is a join, the threadpool passes the child's local time to its parent thread.

$$\frac{(s, \hbar, t), \mathbb{P}^\uparrow \overset{\iota}{\rightsquigarrow_t} (s', \hbar', t'), \mathbb{P}^{\uparrow\prime} \quad \iota \in \{\texttt{id}, \texttt{cmt}(-)\}}{(\hbar, \eta \uplus \{i \mapsto (s, t, \mathbb{P}^\uparrow)\}) \overset{\iota}{\rightsquigarrow_g} (\hbar', \eta \uplus \{i \mapsto (s', t', \mathbb{P}^{\uparrow\prime})\})} \; single$$

$$\frac{(s, \hbar, t), \mathbb{P}^\uparrow \overset{\texttt{fork}(i', \mathbb{P}'')}{\rightsquigarrow_t} (s', \hbar', t'), \mathbb{P}^{\uparrow\prime}}{(\hbar, \eta \uplus \{i \mapsto (s, t, \mathbb{P}^\uparrow)\}) \overset{\texttt{fork}(i', \mathbb{P}'')}{\rightsquigarrow_g} (\hbar', \eta \uplus \{i \mapsto (s', t', \mathbb{P}^{\uparrow\prime}), i' \mapsto (\lambda \texttt{x}.\, 0, t', \mathbb{P}'')\})} \; par$$

$$\frac{(s, \hbar, t), \mathbb{P}^\uparrow \overset{\texttt{join}(i', t'')}{\rightsquigarrow_t} (s', \hbar', t'), \mathbb{P}^{\uparrow\prime}}{(\hbar, \eta \uplus \{i \mapsto (s, t, \mathbb{P}^\uparrow), i' \mapsto (s', t'', \texttt{skip})\}) \overset{\texttt{join}(i', t'')}{\rightsquigarrow_g} (\hbar', \eta \uplus \{i \mapsto (s', t', \mathbb{P}^{\uparrow\prime})\})} \; wait$$

2 Proof of semantics

Lemma 2.1. A history cannot be overwritten, i.e. $\forall \hbar, \hbar', l, t.\, (\hbar, -) \overset{-}{\rightsquigarrow}_g (\hbar', -) \implies \hbar(l)(t) \subseteq \hbar'(l)(t)$

Proof. From the `allowcommit`. □

Lemma 2.2. For any time, it can have at most one write event and, if it has then no read events happen in the same time. This is $\forall \hbar, l, t.\, (-, \mathtt{w}, -) \in \hbar(l)(t) \implies \hbar(l)(t) = \{(-, \mathtt{w}, -)\}$

Proof. From the `allowcommit`. □

Lemma 2.3. All the reads of a transaction happen before all the writes. This is $\forall \hbar, l, l', t, t', \alpha.\, \hbar(l)(t) = (-, \mathtt{r}, \alpha) \wedge \hbar(l')(t') = (-, \mathtt{w}, \alpha) \implies t < t'$.

Proof. From the semantics that $t_s < t_e$. □

Lemma 2.4. All the reads of a transaction happen in the same time, so do all the writes. This is $\forall \hbar, l, l', t, t', \alpha, o \in \{\mathtt{r}, \mathtt{w}\}\,.\, \hbar(l)(t) = \hbar(l')(t') = (-, o, \alpha) \implies t = t'$.

Proof. From the *commit*. □

First we define session order `so`, or program order.

$$
\begin{array}{rcl}
lc \in \mathtt{LastCommit} & \triangleq & \mathtt{ThreadID} \rightharpoonup \mathcal{P}(\mathtt{TransID}) \\
sessionOrder^0(\hbar_{init}, \eta_{init}) & \triangleq & \{(\emptyset, \emptyset, \hbar_{init}, \eta_{init}, \emptyset)\} \\
sessionOrder^n(\hbar_{init}, \eta_{init}) & \triangleq & \left\{ (\mathcal{T}, \mathtt{so}, \hbar, \eta \uplus \{i \mapsto -\}, lc) \middle| \begin{array}{l}
\exists \mathcal{T}', \mathtt{so}', \hbar', \eta', lc', \iota. \\
(\mathcal{T}', \mathtt{so}', \hbar', \eta' \uplus \{i \mapsto -\}, lc') \in \\
\qquad sessionOrder^{n-1}(\eta_{init}, \hbar_{init}) \wedge \\
(\hbar', \eta' \uplus \{i \mapsto -\}) \overset{\iota}{\rightsquigarrow}_g (\hbar, \eta \uplus \{i \mapsto -\}) \wedge \\
\iota = \mathtt{id} \implies \\
\left(\mathcal{T} = \mathcal{T}' \wedge \mathtt{so} = \mathtt{so}' \wedge \eta = \eta' \wedge lc = lc'\right) \wedge \\
\exists \alpha.\, \iota = \mathtt{cmt}(\alpha) \implies \\
\begin{pmatrix} \mathcal{T} = \mathcal{T}' \uplus \{\alpha\} \wedge \\ \mathtt{so} = \mathtt{so}' \uplus \{(\alpha', \alpha) | \alpha' \in lc'(i)\} \wedge \\ \eta = \eta' \wedge lc = lc'[i \mapsto \{\alpha\}] \end{pmatrix} \wedge \\
\exists i'.\, \iota = \mathtt{fork}(i', -) \implies \\
\begin{pmatrix} \mathcal{T} = \mathcal{T}' \wedge \mathtt{so} = \mathtt{so}' \wedge \eta = \eta' \uplus \{i' \mapsto -\} \wedge \\ lc = lc' \uplus \{i' \mapsto lc'(i)\} \end{pmatrix} \wedge \\
\exists i''.\, \iota = \mathtt{join}(i'', -) \implies \\
\begin{pmatrix} \mathcal{T} = \mathcal{T}' \wedge \mathtt{so} = \mathtt{so}' \wedge \eta = \eta' \uplus \{i'' \mapsto -\} \wedge \\ lc = lc'[i \mapsto lc'(i) \uplus lc'(i'')] \setminus \{i'' \mapsto -\} \end{pmatrix} \wedge
\end{array} \right\} \\
histories(\hbar_{init}, \eta_{init}) & \triangleq & \left\{ (\mathcal{T}, \mathtt{so}, \hbar) \middle| (\mathcal{T}, \mathtt{so}, \hbar, -, -) \in \bigcup_{n \in \mathbb{N}} sessionOrder^n(\hbar_{init}, \eta_{init}) \right\}
\end{array}
$$

Lemma 2.5. Transactions ordered by session order `so` appear in the same order in the time stamp heap, i.e. $\forall \hbar_{init}, \eta_{init}, (\mathcal{T}, \mathtt{so}, \hbar) \in histories(\hbar_{init}, \eta_{init}), \alpha, \alpha' \in \mathcal{T}, l.\, \exists t, ts'.\, \hbar(l)(t) = (-, -, \alpha) \wedge \hbar(l)(t') = (-, -, \alpha') \implies t < t'$

Proof. From the definition of session order and the semantics, especially *commit*, *seqnoskip*, *par* and *wait*. □

Now we need to recover `ww`, `wr` and `rw` from $\hbar$. We need to prove that there is no circle that does not contain contain adjacent `rw`. By contradiction, if such circle exists, we can prove that the time goes backward in the circle.

$$
\begin{array}{rcl}
graph(\mathcal{T}, \mathtt{so}, \hbar) & \triangleq & (\mathcal{T}, \mathtt{so}, \mathtt{wr}, \mathtt{ww}, \mathtt{rw}) \\
& & \mathtt{where}\ \forall \alpha, \alpha' \in \mathcal{T}.\, \alpha \neq \alpha' \wedge \exists l, t, t' > t. \\
& & (\alpha, \alpha') \in \mathtt{wr} \implies t = \max(\{t'' | t'' < t\}), v.\, \hbar(l)(t) = (-, \mathtt{w}, \alpha) \wedge \hbar(l)(t') = (-, \mathtt{r}, \alpha') \wedge \\
& & (\alpha, \alpha') \in \mathtt{ww} \implies \hbar(l)(t) = (-, \mathtt{w}, \alpha) \wedge \hbar(l)(t') = (-, \mathtt{w}, \alpha') \wedge \\
& & (\alpha, \alpha') \in \mathtt{rw} \implies \hbar(l)(t) = (-, \mathtt{r}, \alpha) \wedge \hbar(l)(t') = (-, \mathtt{w}, \alpha')
\end{array}
$$

Lemma 2.6. Given threes transactions, α_1, α_2 and α_3, if there is a order between α_1 and α_2, so does α_2 and α_3, and at least one of these two orders are not read-write relation, then the time, associated with the events that determines the orders, strictly increases. The ABOVE NOT REALLY DESCRIBE THE PROPERTY!!! BUT LEAVE IT FOR NOW. BECAUSE MIGHT CHANGE THE PROOF STRATEGY.

This is, given $(\mathcal{T}, \mathtt{so}, \mathtt{wr}, \mathtt{ww}, \mathtt{rw})$ generated from $\hbar$, it satisfies the following property:

$$
\begin{array}{l}
\forall \alpha_1, \alpha_2, \alpha_3 \in \mathcal{T}. \exists \iota, \iota' \in \{\mathtt{so}, \mathtt{wr}, \mathtt{ww}, \mathtt{rw}\}, l_1, l_2, l_3, t_1, t_2, t_3, o_1, o_2, o_3. (\alpha_1, \alpha_2) \in \iota \wedge (\alpha_2, \alpha_3) \in \iota' \wedge \\
\hbar(l_1)(t_1) = (-, o_1, \alpha_1) \wedge \hbar(l_2)(t_2) = (-, o_2, t_2) \wedge \hbar(l_3)(t_3) = (-, o_3, t_3) \wedge t_1 < t_2 < t_3 \wedge \\
\iota = \mathtt{so} \wedge \iota = \mathtt{so} \implies l_2 = l_3 \wedge o_1, o_2, o_3 \in \{\mathtt{w}, \mathtt{r}\} \wedge \\
\iota = \mathtt{so} \wedge \iota = \mathtt{ww} \implies l_2 = l_3 \wedge o_1 \in \{\mathtt{w}, \mathtt{r}\} \wedge o_2 = o_3 = \mathtt{w} \wedge \\
\iota = \mathtt{so} \wedge \iota = \mathtt{wr} \implies l_2 = l_3 \wedge o_1 \in \{\mathtt{w}, \mathtt{r}\} \wedge o_2 = \mathtt{w} \wedge o_3 = \mathtt{r} \wedge \\
\iota = \mathtt{so} \wedge \iota = \mathtt{rw} \implies l_2 = l_3 \wedge o_1 \in \{\mathtt{w}, \mathtt{r}\} \wedge o_2 = \mathtt{r} \wedge o_3 = \mathtt{w} \wedge \\
\iota = \mathtt{ww} \wedge \iota = \mathtt{so} \implies l_1 = l_2 \wedge o_1 = o_2 = \mathtt{w} \wedge o_3 \in \{\mathtt{w}, \mathtt{r}\} \wedge \\
\iota = \mathtt{ww} \wedge \iota = \mathtt{ww} \implies l_1 = l_2 \wedge o_1 = o_2 = o_3 = \mathtt{w} \wedge \\
\iota = \mathtt{ww} \wedge \iota = \mathtt{wr} \implies l_1 = l_2 \wedge o_1 = o_2 = \mathtt{w} \wedge o_3 = \mathtt{r} \wedge \\
\iota = \mathtt{ww} \wedge \iota = \mathtt{rw} \implies l_2 = l_3 \wedge o_1 = \mathtt{w} \wedge o_2 = \mathtt{r} \wedge o_3 \mathtt{w} \wedge \\
\iota = \mathtt{wr} \wedge \iota = \mathtt{so} \implies l_1 = l_2 \wedge o_1 = \mathtt{w} \wedge o_2 = \mathtt{r} \wedge o_3 \in \{\mathtt{w}, \mathtt{r}\} \wedge \\
\iota = \mathtt{wr} \wedge \iota = \mathtt{ww} \implies l_2 = l_3 \wedge o_1 = o_2 = o_3 = \mathtt{w} \wedge \\
\iota = \mathtt{wr} \wedge \iota = \mathtt{wr} \implies l_2 = l_3 \wedge o_1 = o_2 = \mathtt{w} \wedge o_3 = \mathtt{r} \\
\iota = \mathtt{wr} \wedge \iota = \mathtt{rw} \implies l_2 = l_3 \wedge o_1 = \mathtt{w} \wedge o_2 = \mathtt{r} \wedge o_3 = \mathtt{w} \wedge \\
\iota = \mathtt{rw} \wedge \iota = \mathtt{so} \implies l_1 = l_2 \wedge o_1 = \mathtt{r} \wedge o_2 = \mathtt{w} \wedge o_3 \in \{\mathtt{w}, \mathtt{r}\} \wedge \\
\iota = \mathtt{rw} \wedge \iota = \mathtt{ww} \implies l_2 = l_3 \wedge o_1 = \mathtt{r} \wedge o_2 = o_3 = \mathtt{w} \wedge \\
\iota = \mathtt{rw} \wedge \iota = \mathtt{wr} \implies l_2 = l_3 \wedge o_1 = \mathtt{r} \wedge o_2 = \mathtt{w} \wedge o_3 = \mathtt{r}
\end{array}
$$

3 blablabla

$$P, Q \in \texttt{GlobalAssertion} \triangleq \texttt{False} \mid \texttt{True} \mid \texttt{emp} \mid x \mapsto v \mid P * Q \mid P \wedge Q \mid P \vee Q \mid \exists x.\, P \mid P \implies Q$$

$$p, q \in \texttt{LocalAssertion} \triangleq \texttt{False} \mid \texttt{True} \mid \texttt{emp} \mid x \mapsto_w v \mid x \mapsto_r v \mid p * q \mid p \wedge q \mid p \vee q \mid \exists x.\, p \mid p \implies q$$

$$\begin{aligned} a \in \texttt{Action} &\triangleq \{p \rightsquigarrow q \mid p, q \in \texttt{LocalAssertion} \wedge p \texttt{ has no write tag}\} \\ a \in \texttt{Action} &\triangleq p \rightsquigarrow q \mid a; \texttt{Action} \end{aligned}$$

Logic expression

$$[\![e]\!]_\iota$$

Assume there are special logical variables, ranging $l_1, l_2, \ldots$, that point to the corresponding locations in the global heap.

$$\begin{aligned} [\![\texttt{False}]\!]_\iota &\triangleq \emptyset \\ [\![\texttt{True}]\!]_\iota &\triangleq \texttt{Heap} \\ [\![x \mapsto v]\!]_\iota &\triangleq \{h \mid \exists t.\, h([\![x]\!]_\iota) = ([\![v]\!]_\iota, t)\} \\ [\![P * Q]\!]_\iota &\triangleq \{h_P \uplus h_Q \mid h_P \in [\![P]\!]_\iota \wedge h_Q \in [\![Q]\!]_\iota\} \\ [\![P \wedge Q]\!]_\iota &\triangleq \{h \mid h \in [\![P]\!]_\iota \wedge h \in [\![Q]\!]_\iota\} \\ [\![P \vee Q]\!]_\iota &\triangleq \{h \mid h \in [\![P]\!]_\iota \vee h \in [\![Q]\!]_\iota\} \\ [\![\exists x.\, P]\!]_\iota &\triangleq \left\{h \mid \exists v.\, h \in [\![P]\!]_{\iota[x \mapsto v]}\right\} \\ [\![P \implies Q]\!]_\iota &\triangleq \{h \mid h \in [\![P]\!]_\iota \implies h \in [\![Q]\!]_\iota\} \end{aligned}$$

$$\begin{aligned} [\![\texttt{False}]\!]_{s,\iota} &\triangleq \emptyset \\ [\![\texttt{True}]\!]_{s,\iota} &\triangleq \texttt{Heap} \times \texttt{Heap} \\ [\![\texttt{emp}]\!]_{s,\iota} &\triangleq \{(\emptyset, \emptyset)\} \\ [\![x \mapsto_r v]\!]_{s,\iota} &\triangleq \left\{(h_s, h_e) \mid \exists t.\, h_s([\![x]\!]_{s,\iota}) = ([\![v]\!]_{s,\iota}, t) \wedge h_e = ([\![v]\!]_{s,\iota}, t)\right\} \\ [\![x \mapsto_w v]\!]_{s,\iota} &\triangleq \left\{(h_s, h_e) \mid \exists v_s, t_s, t_e.\, h_s([\![x]\!]_{s,\iota}) = (v_s, t_s) \wedge h_e = ([\![v]\!]_{s,\iota}, t_e) \wedge t_s < t_e\right\} \\ [\![p * q]\!]_{s,\iota} &\triangleq \left\{(h_{ps} \uplus h_{qs}, h_{pe} \uplus h_{qe}) \mid (h_{ps}, h_{pe}) \in [\![p]\!]_{s,\iota} \wedge (h_{qs}, h_{qe}) \in [\![q]\!]_{s,\iota}\right\} \\ [\![p \wedge q]\!]_{s,\iota} &\triangleq \left\{(h_s, h_e) \mid (h_s, h_e) \in [\![p]\!]_{s,\iota} \wedge (h_s, h_e) \in [\![q]\!]_{s,\iota}\right\} \\ [\![p \vee q]\!]_{s,\iota} &\triangleq \left\{(h_s, h_e) \mid (h_s, h_e) \in [\![p]\!]_{s,\iota} \vee (h_s, h_e) \in [\![q]\!]_{s,\iota}\right\} \\ [\![\exists x.\, p]\!]_{s,\iota} &\triangleq \left\{(h_s, h_e) \mid \exists v.\, (h_s, h_e) \in [\![p]\!]_{s,\iota[x \mapsto v]}\right\} \\ [\![p \implies q]\!]_{s,\iota} &\triangleq \left\{(h_s, h_e) \mid (h_s, h_e) \in [\![p]\!]_{s,\iota} \implies (h_s, h_e) \in [\![q]\!]_{s,\iota}\right\} \end{aligned}$$

$$\begin{aligned} \texttt{True} \mathbin{\hat{*}} p &\triangleq p \\ \texttt{False} \mathbin{\hat{*}} - &\triangleq \texttt{False} \\ x \mapsto_r v \mathbin{\hat{*}} x \mapsto_r v &\triangleq x \mapsto_r v \\ x \mapsto_r - \mathbin{\hat{*}} x \mapsto_w v &\triangleq x \mapsto_w v \\ (p_1 * q_1) \mathbin{\hat{*}} (p_2 * q_2) &\triangleq (p_1 \mathbin{\hat{*}} p_2) * (q_1 \mathbin{\hat{*}} q_2) \\ (p \wedge q) \mathbin{\hat{*}} r &\triangleq (p \mathbin{\hat{*}} r) \wedge (q \mathbin{\hat{*}} r) \\ (p \vee q) \mathbin{\hat{*}} r &\triangleq (p \mathbin{\hat{*}} r) \vee (q \mathbin{\hat{*}} r) \\ (p \vee q) \mathbin{\hat{*}} r &\triangleq (p \mathbin{\hat{*}} r) \vee (q \mathbin{\hat{*}} r) \\ \exists x.\, p \mathbin{\hat{*}} p &\triangleq \exists x.\, (p \mathbin{\hat{*}} r) \\ (p \implies q) \mathbin{\hat{*}} r &\triangleq (p \mathbin{\hat{*}} r) \implies (q \mathbin{\hat{*}} r) \\ \vdash p, p' \texttt{ agree} &\iff \exists r, r', r'' \neq \texttt{False}.\, p * r \mathbin{\hat{*}} p' * r' = r'' \\ R \vdash p, q \texttt{ merge } q' &\iff \forall p_R \rightsquigarrow q_R \in R. \vdash p, p_R \texttt{ agree} \wedge \vdash q, q_R \texttt{ agree} \\ &\quad \implies (\exists r, r', r_R.\, (q * r) \mathbin{\hat{*}} (q_R * r_R) \implies q' * r' \wedge q \implies q') \\ R \vdash q \texttt{ stable} &\iff \forall p_R \rightsquigarrow q_R \in R. \exists p.\, q \overset{\texttt{retag}\ r}{\Longrightarrow} p \wedge \vdash p, p_R \texttt{ agree} \wedge \\ &\quad \exists r, r_R.\, (q * r) \mathbin{\hat{*}} (q_R * r_R) = q * r \end{aligned}$$

$$\frac{P \overset{\texttt{tag}\ r}{\Longrightarrow} p \quad \vdash \{p\}\ \mathbb{C}\ \{q\} \quad (p \rightsquigarrow q) \in G^* \qquad R \vdash p, q \texttt{ merge } q' \quad R \vdash q' \texttt{ stable} \quad q' \overset{\texttt{notag}}{\Longrightarrow} Q}{R, G \vdash \{P\}\ [\mathbb{C}]\ \{Q\}}\ \textit{commit}$$

$$\frac{R, G \vdash \{P \wedge \mathbb{B}\}\ \mathbb{P}_1\ \{Q\} \qquad R, G \vdash \{P \wedge \neg\mathbb{B}\}\ \mathbb{P}_2\ \{Q\}}{R, G \vdash \{P\}\ \texttt{if}\,(\mathbb{B})\ \mathbb{P}_1\ \texttt{else}\ \mathbb{P}_2\ \{Q\}}\ \textit{choice}$$

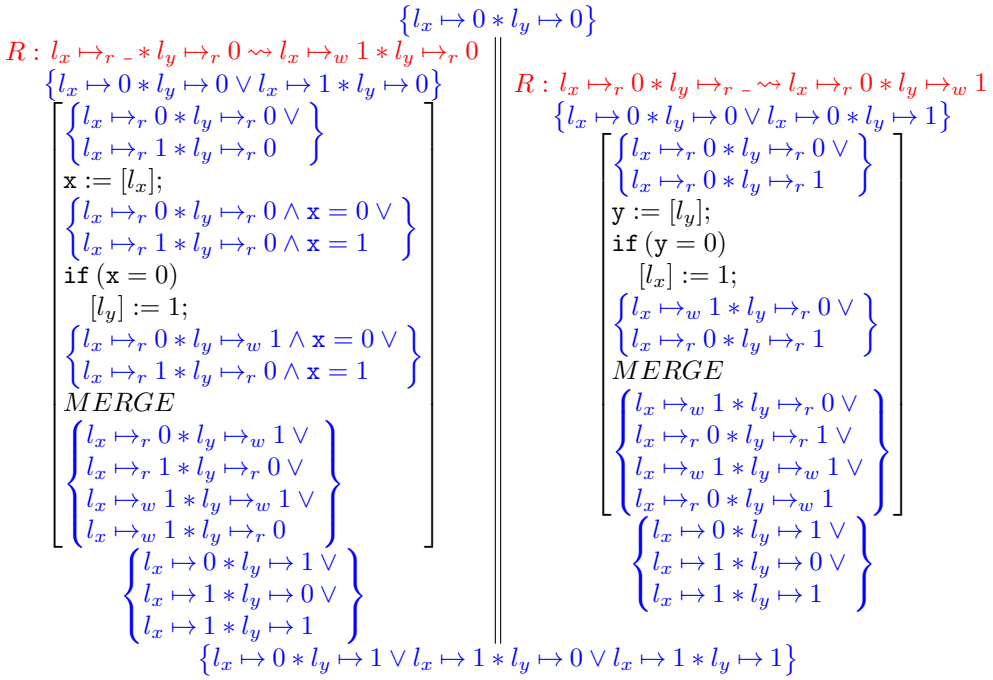
$$\frac{R, G \vdash \{P\}\ \mathbb{P}_1\ \{Q'\} \qquad Q' \implies P' \qquad R, G \vdash \{P'\}\ \mathbb{P}_2\ \{Q\}}{R, G \vdash \{P\}\ \mathbb{P}_1 ; \mathbb{P}_2\ \{Q\}}\ \textit{seq}$$

$$\frac{\begin{array}{l} R \cup G_2, G_1 \vdash \{P_1\}\ \mathbb{P}_1\ \{Q_1\} \qquad R \cup G_1, G_2 \vdash \{P_2\}\ \mathbb{P}_2\ \{Q_2\} \\ R \cup G_2 \vdash P_1\ \texttt{stable} \qquad R \cup G_1 \vdash P_2\ \texttt{stable} \end{array}}{R, G_1 \cup G_2 \vdash \{P_1 * P_2\}\ \mathbb{P}_1 \parallel \mathbb{P}_2\ \{Q_1 * Q_2\}}\ \textit{concur}$$

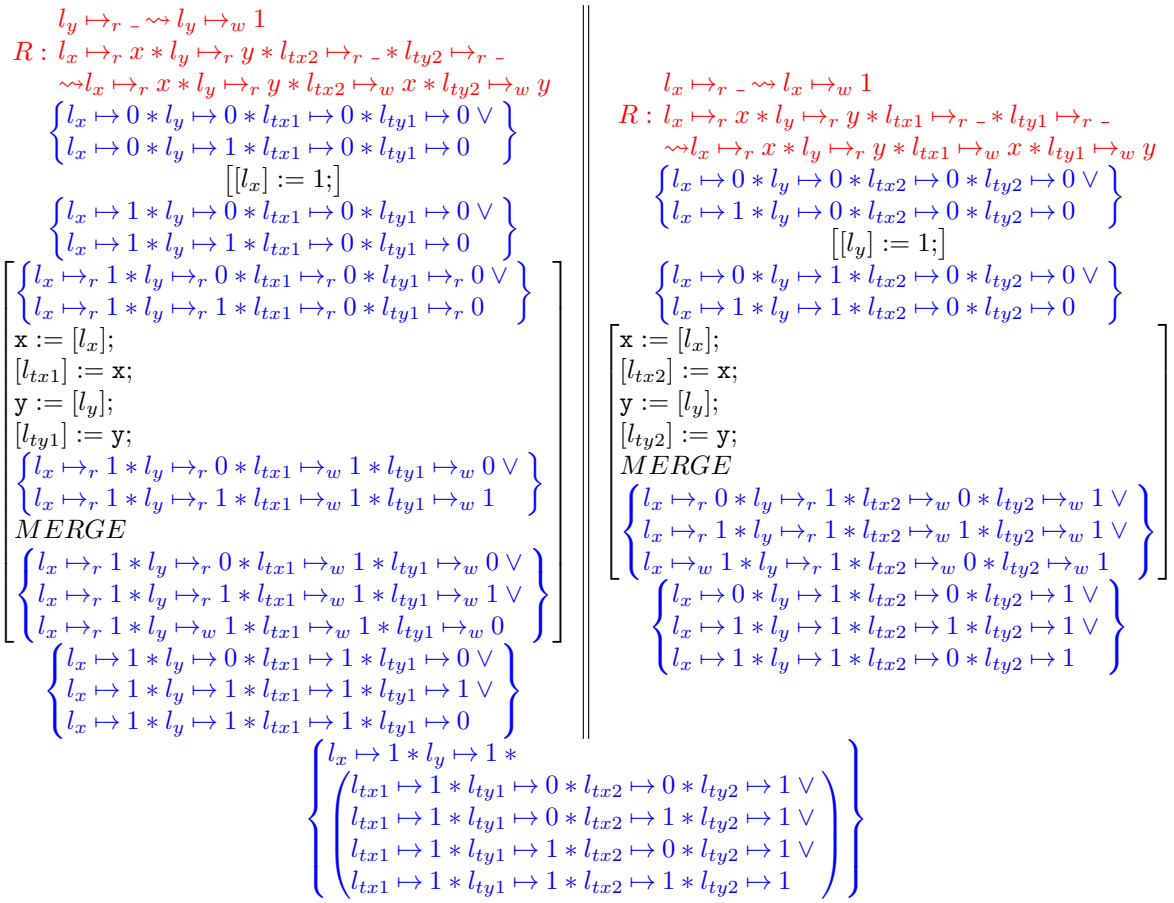
$$\frac{R, G \vdash \{P \wedge \mathbb{B}\}\ \mathbb{P}\ \{P\}}{R, G \vdash \{P\}\ \texttt{while}\,(\mathbb{B})\ \mathbb{P}\ \{P \wedge \neg\mathbb{B}\}}\ \textit{repeat}$$

4 blablabla

We use $l[-]$ to denote a location in either global or local heap.

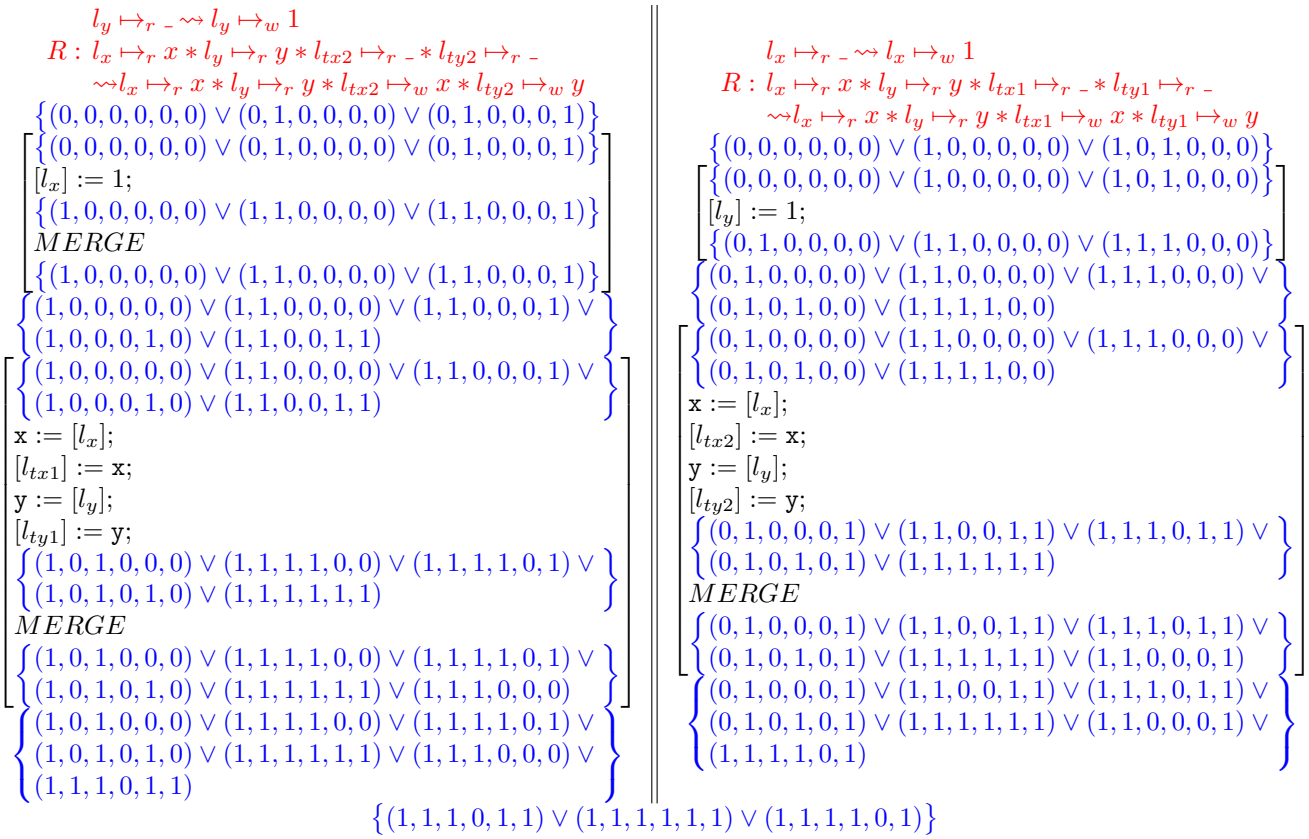


***The second rely might be useful in term of killing the possible states of other thread, even though it is irrelevant for the current thread. About the problem, an ugly solution is: if the resource appear in rely, current thread should keep it whether uses it or not. However this solution is against the idea of separation logic, i.e. we do not need to take care of those resource untouched.



Above, the 1 0 0 1 case will never happen. This is called long fork in snapshot isolation, for a single machine snapshot isolation, it should not happen.

We use $(-,-,-,-,-,-)$ to refer x,y,tx1,ty1,tx2,ty2.



$$\begin{bmatrix}[l_x] := 1;\\ [l_z] := 1;\end{bmatrix} \parallel \begin{bmatrix}[l_y] := 2;\\ [l_z] := 2;\end{bmatrix}$$