GSP-Calculus

Hernán Melgratti[1,2] and Christian Roldán[1]

[1] Departamento de Computación, FCEyN, Universidad de Buenos Aires.
[2] CONICET.

Abstract.

1 Global Sequence Protocol Language

1.1 Syntax

We assume the following countable sets of vertice names $\mathbb{V}$ ranged over by $v;v_0;v_1,\ldots$; We shall use α, β to denote sequence of updates $\mathbb{U}$ ranged over by $u^v,u_0^{v_0},u_1^{v_1},\ldots$; Let $\mathcal{U}$=$u_0^{v_0}\cdot u_1^{v_1}\cdot,\ldots,\cdot u_n^{v_n}$. For the sake of clarity, we shall only use vertices as update's decoration whenever it would be necessary.

Let ρ, σ be sequence defined as $\varepsilon \mid [\alpha]\cdot\rho$. We use ε to denote the empty sequence, i.e., the sequence of length 0.

Let ρ and α be sequence, we write $\setminus$ as relative complement of ρ in α, i.e. ,$\rho \setminus \alpha = [u_i \mid u_i \in \rho \wedge u_i \notin \alpha]$.

Definition 1.1 (Global Sequence Protocol Language). *The set of tgsp-calculus expressions is defined by the following syntax:*

$$
\begin{array}{lll}
(\textsc{system}) & N := C \parallel S & \\
(\textsc{client}) & C := 0 \mid \{P,k,\rho,\alpha,\sigma,j\} \mid C \parallel C & \textit{where } k \textit{ and } j \in \mathbb{N} \\
(\textsc{state}) & S := \varepsilon \mid S\cdot\rho & \\
(\textsc{program}) & P := \textit{update(u);}P \mid \textit{let } x = \textit{read}(v);P \mid \textit{pull();}P \mid \textit{push();}P \mid & \\
 & \quad \textit{let } x = \textit{confirmed}();P \mid \textit{while}(\textit{cond})\ \textbf{\textit{do}}\ P &
\end{array}
$$

We model the system N as several clients interacting with a message queue S. This message queue will represent the program state. We formalize an i^{th} client as a tuple $\{P,k,\rho,\alpha,\sigma,j\}_i$ where P, called Program, is a set of programming primitives that allow programmer to interact with the distributed store by invoking of actions on objects, a index k which denotes the maximum index of update that reading in S, a sequence of updates sent but unconfirmed their reception ρ, a transaction buffer α, a sequence of all the updates sent by the client to the RTOB σ and finally j which represents a amount of received updates.

1.2 Operational Semantics

The operational semantics of *tgsp-calculus* is defined by a labeled transition system over well-formed terms, up-to the structural congruence.

The labeled transition system considers the following actions:

$$\alpha ::= \tau \mid \mathit{read}(r) \mid \mathit{update}(u) \mid \mathit{pull}() \mid \mathit{push}() \mid \mathit{confirmed}()$$

(T-READ)
$$\frac{\mathit{rvalue}(r,\mathit{flatten}(S[0..k-1]\cdot\rho)\cdot\alpha) = v}{\{\mathit{let}\ x = \mathit{read}(r); P,k,\rho,\alpha,\sigma,j\}_i \parallel C \parallel S \xrightarrow{\mathit{read}(r)} \{P[x\mapsto v],k,\rho,\alpha,\sigma,j\}_i \parallel C \parallel S}$$

(T-UPDATE)
$$\{\mathit{update}(u); P,k,\rho,\alpha,\sigma,j\}_i \parallel C \parallel S \xrightarrow{\mathit{update}(u)} \{P,k,\rho,\alpha\ \cdot u,\sigma,j\}_i \parallel C \parallel S$$

(T-PUSH)
$$\{\mathit{push}(); P,k,\rho,\alpha,\sigma,j\}_i \parallel C \parallel S \xrightarrow{\mathit{push}()} \{P,k,\rho\cdot[\alpha],\varepsilon,\sigma\cdot[\alpha],j\}_i \parallel C \parallel S$$

(T-PULL)
$$\{\mathit{pull}(); P,k,\rho,\alpha,\sigma,j\}_i \parallel C \parallel S \xrightarrow{\mathit{pull}()} \{P,k+j,\rho\setminus S[k..k+j],\alpha,\sigma,0\}_i \parallel C \parallel S$$

(T-CONFIRMED)
$$\{\mathit{let}\ x = \mathit{confirmed}(); P,k,\rho,\alpha,\sigma,j\}_i \parallel C \parallel S \xrightarrow{\mathit{confirmed}()} \{P[x\mapsto \rho\neq[]\vee\ \alpha\neq\varepsilon],k,\rho,\alpha,\sigma,j\}_i \parallel C \parallel S$$

(T-WHILE-TRUE)
$$\frac{\neg(\mathit{cond}\downarrow) = \mathit{false}}{\{\mathit{while}(\mathit{cond})\ \mathbf{do}\ P,k,\rho,\alpha,\sigma,j\}_i \parallel C \parallel S \xrightarrow{\tau} \{P;\mathit{while}(\mathit{cond})\ \mathbf{do}\ P,k,\rho,\alpha,\sigma,j+1\}_i \parallel C \parallel S}$$

(T-WHILE-FALSE)
$$\frac{\neg(\mathit{cond}\downarrow) = \mathit{true}}{\{\mathit{while}(\mathit{cond})\ \mathbf{do}\ P,k,\rho,\alpha,\sigma,j\}_i \parallel C \parallel S \xrightarrow{\tau} \{0,k,\rho,\alpha,\sigma,j+1\}_i \parallel C \parallel S}$$

(T-RECEIVE)
$$\frac{k+j+1 \leq |S|}{\{P,k,\rho,\alpha,\sigma,j\} \parallel C \parallel S \xrightarrow{\tau} \{P,k,\rho,\alpha,\sigma,j+1\}_i \parallel C \parallel S}$$

(T-PROCESS)
$$\{P,k,\rho,\alpha,[\beta]\cdot\sigma,j\} \parallel C \parallel S \xrightarrow{\tau} \{P,k,\rho,\alpha,\sigma,j\} \parallel C \parallel S\cdot\beta$$

The meaning of τ is standard. A *tgsp-calculus* program is given by a Labeled Transition System (LTS) before introducing. Transitions for systems are labeled by actions α as usual.

Rule (T-READ) states that a client i can perform an action $\mathit{read}(r)$, substituting each occurrence of x in P with the value v. Notice that v is got from the queue message and the client's internal queues. Rule (T-UPDATE) describes effects performed by an

update operation. The update u is saving in the temporal queue α. After (T-PUSH) the content of the transaction queue α is moved to the pending queue, ρ, and the sent queue, σ. Rule (T-PULL) increases the amount of updates that client i can read in the message queue S. This increase is given by the amount of update receiving.The pending queue removes those updates which belong between index k and the amount of update receiving j in S. Besides, j is restarted in 0. (T-CONFIRMED) sets x in false whether updates are in pending queue or transaction queue, true in otherwise. Rule (T-WHILE-TRUE) and (T-WHILE-FALSE) are standard. Rule (T-RECEIVE) states that a counter j is incremented (by 1). Finally, the last one rule, (T-PROCESS), moves updates from sent queue to the message queue.

Notation. Let f and g be a partial function, we define the update operator $\_[\_]$ such that $dom(f[g]) = dom(f) \cup dom(g)$ and

$$f[g](x) = \begin{cases} f(x) \text{ if } x \notin dom(g) \wedge x \in dom(f) \\ g(x) \text{ if } x \in dom(g) \\ \bot \quad \text{Otherwise} \end{cases}$$

We write $[x_1 \mapsto y_1, \ldots, x_n \mapsto y_n]$ for the partial function f such that $dom(f) = \{x_1, \ldots, x_n\}$ and $f(x_i) = y_i$; $A \setminus B$ to denote the usual difference of sets.

2 Robust Streaming

Let *State* and *Delta* be abstract types, *GSSegment* the set operation over sets $Delta \times \mathcal{N} \to \mathbb{N}$, *GSPrefix* the set over sets $State \times \mathcal{N} \to \mathbb{N}$ and *Round* the set over sets $\mathcal{N} \times \mathbb{N} \times Delta$; θ is a fresh element of *GSSegment*.

Furthermore, we assume the following parametric functions defined over the abstract data types above borrowed from[1]:

const	*initialstate*	:*State*
function	*read*	: $Read \times State \to Value$
function	*apply*	: $State \times Delta^* \to State$
const	*emptydelta*	: *Delta*
function	*append*	: $Delta \times Update \to Delta$
function	*reduce*	: $Delta^* \to Delta$

Let *State* be a set of state ranged over by s, s_1, s_2, ...; We shall use δ to denote sequence of Delta ranged over by d, d_0, d_1 ...; Let ρ, σ be sequence defined:

ρ,σ := ε | [δ]· ρ

We assume the following countable sets of persisted state *GSPrefix* ranged over by ps, ps_1, ...;

Let in_s be a partial function in $\mathcal{N} \to \mathbb{N} \times Delta$

Let out_s be a partial function in $\mathcal{N} \to GSSegment^* \cup GSPrefix$

2.1 Auxiliar Function

$append :: GSSegment\ x\ Round^* \to State$
$append(<\delta, mr>, \varepsilon) = <\delta, mr>$
$append(<\delta, mr>, <i, n_0, \delta_0>:rs) = append(<\textbf{\textit{reduce}}(\delta \cdot \delta_0 \cdot \varepsilon), mr[b_0 \mapsto n_0]>, rs)$

$apply :: GSPrefix\ x\ GSSegment \to GSPrefix$
$apply(<ps, mr>, <\delta, mr'>) = <\textbf{\textit{apply}}(ps, \delta \cdot \varepsilon), mr[mr']>$

$receivedrounds :: (\mathcal{N} \to <\mathbb{N} \times Delta>^*) \to Round^*$
$receivedrounds(\bot) = \varepsilon$
$receivedrounds(b \to <n_0,\ \delta_0> \cdot\ f) = <b,\ n_0,\ \delta_0> \cdot receivedrounds(f)$

$curstate :: State\ x\ Round^*\ x\ Delta\ x\ Delta \to State$
$curstate(ps, p, pb, tb) = \textbf{\textit{apply}}(ps, getdeltas(p) \cdot pb \cdot tb)$

$getdeltas :: Round^* \to Delta^*$
$getdeltas(\varepsilon) = \varepsilon$
$getdeltas(<n_0, \delta_0> \cdot \delta) = \delta_0 \cdot \text{getdeltas}(\delta)$

Definition 2.1 (GSP). *The syntax of clients and server is given by the following grammar*

$$
\begin{array}{ll}
(\text{SYSTEM}) & N ::= 0 \mid S \parallel C \\
(\text{SERVER}) & S ::= 0 \mid \{ps, in_s, out_s\} \\
(\text{CLIENTS}) & C ::= \emptyset \mid \{P, <State, \rho, \delta, \delta, GSSegment \cup GSPrefix, \mathbb{N}, in_c, out_c>\} \mid C \parallel C \\
(\text{PROGRAM}) & P ::= 0 \mid let\ x = read(r); P \mid update(u); P \mid push(); P \mid pull(); P
\end{array}
$$

E.$k \in State$; E.$p \in \rho$; E.pb, E.$tb \in \delta$; E.$n \in \mathbb{N}$.

2.2 Operational Semantics

The operational semantics of *gsp-calculus* is defined by a labeled transition system over well-formed terms, up-to the structural congruence.

The labeled transition system considers the following actions:

$$\alpha ::= \tau \mid read(r) \mid update(u) \mid pull() \mid push() \mid confirmed()$$

SERVER

(DROP_CONN)

$$\frac{b_i \in in_s \qquad b_i \in out_s}{\{ps, in_s, out_s\} \xmapsto{\dagger(b_i)} \{ps, in_s \setminus b_i, out_s \setminus b_i\}}$$

(CRASH_AND_RECOVER)

$$\{ps, in_s, out_s\} \overset{\ddagger}{\mapsto} \{ps, \bot, \bot\}$$

(BATCH)

$$\frac{ps' = apply(ps, d) \qquad d = append(\theta, rs) \qquad rs = receivedRounds(in_s)}{\{ps, in_s, out_s\} \xrightarrow{\tau} \{ps', in_s{}^p s', out_s\}}$$

(ACCEPT_CONN)

$$\frac{b_i \notin in_s \qquad b_i \notin out_s}{\{ps, in_s, out_s\} \xmapsto{\oplus(b_i)} \{ps, in_s, out_s[b_i \mapsto ps]\}}$$

COMMUNICATION

(COMM SERVER-CLIENT)

$$\frac{out_s(i) = gs \cdot gss}{\{ps, in_s, out_s\} \| \{P, E\}_i \xrightarrow{\tau} \{ps, in_s, out_s[i \mapsto gss]\} \| \{P, E[in_c \mapsto E.in_c \cup \{gs\}]\}_i}$$

(COMM CLIENT-SERVER)

$$\frac{out_c = <i, n_0, \delta_0> \cdot rs}{\{ps, in_s, out_s\} \| \{P, E\}_i \xrightarrow{\tau} \{ps, in_s[i \mapsto <i, n_0, \delta_0>], out_s\} \| \{P, E[out_c \mapsto rs]\}_i}$$

CLIENTS

(READ)

$$\frac{\textbf{\textit{read}}(r, curstate(k,p,pb,tb)) = v}{\{let\ x = read(r); P, E\}_i \xrightarrow{read(r)} \{P[x \mapsto v], E\}_i}$$

(UPDATE)

$$\{update(u); P, E\}_i \xrightarrow{update(u)} \{P, E[tb \mapsto \textbf{\textit{append}}(E.tb, u)]\}_i$$

(PUSH)

$$\{push(); P, E\}_i \xrightarrow{push()} \{P, E[pb \mapsto \textbf{\textit{reduce}}(E.pb \cdot E.tb); tb \mapsto \varepsilon; n \mapsto E.n + 1]\}_i$$

(PULL)

$$\frac{E.in_c \neq \emptyset \qquad E.out_c \neq \emptyset \qquad |E.rb| > 0}{\{pull(); P, E\}_i \xrightarrow{pull()} \{P, E[k \mapsto \textbf{\textit{reducestate}}(E.k, E.rb); rb \mapsto \varepsilon]\}_i}$$

(CONFIRMED-TRUE)

$$\frac{E.p = \varepsilon \qquad E.pb = \varepsilon \qquad E.tb = \varepsilon}{\{let\ x = confirmed(); P, E\}_i \xrightarrow{confirmed()} \{P[x \mapsto \textbf{true}], E\}_i}$$

(CONFIRMED-FALSE)

$$\frac{E.p \neq \varepsilon \;||\; E.pb \neq \varepsilon \;||\; E.tb \neq \varepsilon}{\{let\ x = confirmed(); P, E\}_i \xrightarrow{confirmed()} \{P[x \mapsto \textbf{false}], E\}_i}$$

(SEND)

$$\{P, E\}_i \xrightarrow{send()} \{P, E[p \mapsto E.p \cdot E.pb; pb \mapsto \varepsilon; out_c \mapsto out_c \cdot <i, E.n, E.pb>]\}_i$$

(RECEIVE)

$$\frac{E.in_c = gs_0 \cdot gs_t}{\{P, E\}_i \xrightarrow{receive()} \{P, E[rb \mapsto E.rb \cdot gs_0.gssegment); in_c \mapsto gs_t)]\}_i}$$

(DROP_CONNECTION)

$$\{P, k\}_i E \xrightarrow{\dagger} \{P, E[in_c \mapsto \emptyset; out_c \mapsto \emptyset]\}_i$$

3 Consistency Guarantees

We shall introduce a series of store-level consistency guarantees and then we shall show which are captured by application written in GSP. We start identifying three kinds of relations between actions of update and read:

Session Order relates whatever pair of actions from the same client, indicating the program order. It is a total order on actions.

Visibility relates Updates with Reads. It is used to indicate is if an action of Update is visible for an action of Read.

Arbitration relates Updates with Updates. It is used to resolve update conflicts. It is a total order on actions of update.

We extend the GSP language with a new term which capture the relations amount operation in our system.

$$(\text{ENVIRONMENT}) \quad E ::= \{N, \text{OP}, \text{SO}, \text{VIS}, \text{AR}\}$$

Let E, a new term, where N represents our system introduced in Definition 1.1, OP is a mapping of vertices to actions, SO is a session order relation defined from vertices to relations of vertices $\mathbb{V} \times (\mathbb{V} \times \mathbb{V})$ and VIS,AR are visibility and arbitration relation.

Notation. Given a session order relation SO from client i and a vertice v, we shall write $\text{SO} \rhd_i v$ meaning that $(\mathcal{V}, \mathcal{R}) \rhd_i v = (\mathcal{V} \cup \{v\}, \mathcal{R} \cup \{(x,v)/x \in \mathbb{V}\})$. We shall refer to an update action on the queue message as $S[n]_i$.The arbitration relation AR is defined as $\{(v,w)/\{v \mapsto S[m]_h\} \in \text{OP} \wedge \{w \mapsto S[n]_i\} \in \text{OP} \wedge m < n\}$. A transition $\xrightarrow{\alpha}_i$ denotes the fact that action α is perfomed by client i.

The following operational semantic allow to understand how working the environment when the actions are executed.

(E-READ)

$$\frac{N \xrightarrow{read(r)}_i N' \qquad v \notin dom(\text{OP}) \qquad \text{VIS}' = \text{VIS} \cup \{(x,v)/\{x \mapsto update(u)_h\} \in \text{OP} \wedge u^x \in S[0..k-1] \cdot \rho \cdot [\alpha]\}}{\{N, \text{OP}, \text{SO}, \text{VIS}, \text{AR}\} \xrightarrow{read(r)}_i \{N', \text{OP} \cup \{v \mapsto read(r)\}, \text{SO} \rhd_i v, \text{VIS}', \text{AR}\}}$$

(E-UPDATE)

$$\frac{N \xrightarrow{update(u^v)}_i N' \qquad v \notin dom(\text{OP})}{\{N, \text{OP}, \text{SO}, \text{VIS}, \text{AR}\} \xrightarrow{update(u^v)}_i \{N', \text{OP} \cup \{v \mapsto update(u)\}, \text{SO} \rhd_i v, \text{VIS}, \text{AR}\}}$$

3.1 Ordering Guarantees

We now prove what ordering guarantees are assured by GSP language and what do not.

First, we prove a useful lemma:

Lemma 3.1. *Let u an update action, S a message queue, ρ_i and α_i a pending queue and transaction queue from the client i, if $\{N, \emptyset, \emptyset, \emptyset, \emptyset\} \rightarrow^* \{N, \text{OP}, \text{SO}, \text{VIS}, \text{AR}\}$ then $\{x \mapsto update(u)_i\} \in \text{OP} \Rightarrow u^x \in S[0..k_i - 1] \cdot \rho_i \cdot [\alpha_i]$*

Proof. The proof follows by induction on the length of the derivation $\rightarrow^*$.

- **n=0**. Then OP is $\emptyset$, so that antecedent is false, then the preoposition is true.
- **n=k+1**. Then $\{N, \emptyset, \emptyset, \emptyset, \emptyset\} \rightarrow^n \{N, \text{OP}, \text{SO}, \text{VIS}, \text{AR}\} \rightarrow_i \{x \mapsto update(u)_i\} \in \text{OP} \Rightarrow u^x \in S[0..k_i - 1] \cdot \rho_i \cdot [\alpha_i]$. We proceed by case analysis on the last transition:

- **rule (E-READ)**. As x is an update operation then it must not be v, so that $\{x \mapsto update(u)_i\} \in$ OP, then by inductive hypothesis $u^x \in S[0..k_i-1]\cdot\rho_i\cdot[\alpha_i]$. When $N \xrightarrow{read(r)}_i N'$, k_i, ρ_i,α_i do not change.
- **rule (E-UPDATE)**. There are two possibilities:
  - $x \neq v$. Then we can use inductive hypothesis, so that it is easy to see that if $u^x \in S[0..k_i-1]\cdot\rho_i\cdot[\alpha_i]$ then $u^x \in S[0..k_i-1]\cdot\rho_i\cdot[\alpha_i]\cdot u_t^v$ too.
  - $x = v$. When $N \xrightarrow{update(u^v)}_i N'$, $u^v \in \alpha_i'$ (with α_i' transaction queue in N') because $\alpha_i' = \alpha_i \cdot u^v$. It is immediate to note that $u^x \in S[0..k_i'-1]\cdot\rho_i'\cdot[\alpha_i']$.
- **rule (E-PUSH)**. As OP do not change, then by inductive hypothesis, $u^x \in S[0..k_i-1]\cdot\rho_i\cdot[\alpha_i] \equiv u^x \in S[0..k_i-1]\cdot(\rho_i\cdot[\alpha_i])\cdot\varepsilon$. When $N \xrightarrow{push()}_i N'$, k_i' = k_i, ρ_i' = $\rho_i\cdot[\alpha_i]$ and α_i' = ε.
- **rule (E-PULL)**. As OP do not change, then by inductive hypothesis, $u^x \in S[0..k_i-1]\cdot\rho_i\cdot[\alpha_i]$. We should prove that it is equivalent to $u^x \in S[0..k_i-1+j_i]\cdot\rho_i \setminus S[k_i..k_i+j_i]\cdot[\alpha_i]$. There are two interesting cases to consider:
  - If $u^x \in \rho_i \wedge u^x \notin S[k_i..k_i+j_i]$, then $u^x \in \rho_i$'.
  - If $u^x \in \rho_i \wedge u^x \in S[k_i..k_i+j_i]$, then $u^x \notin \rho_i$' but $u^x \in S[k_i..k_i+j_i]$.

The proof for the remaining cases is by inductive hypothesis because OP, k_i, ρ_i and α_i do not change.

Theorem 3.1 (READ MY WRITES).
Let SO_R *the second component of the relation* SO *and* VIS *a visibility relation, if* $\{N_0,\emptyset,\emptyset,\emptyset,\emptyset\} \to^* \{N, \text{OP}, \text{SO}, \text{VIS}, \text{AR}\}$ *then* $SO_R \cap (\mathbb{U}\times\mathbb{R}) \subseteq$ VIS

Proof. The proof follows by induction on the length of the derivation $\to^*$.

- **n=0**. In particular OP and VIS are $\emptyset$, so that $\emptyset \subseteq \emptyset$.
- **n=k+1**. Then $\{N_0,\emptyset,\emptyset,\emptyset,\emptyset\} \to^n \{N, \text{OP}, \text{SO}, \text{VIS}, \text{AR}\} \xrightarrow{\alpha}_i \{N', \text{OP}', \text{SO}', \text{VIS}', \text{AR}\}$. We proceed by definition:
  - SO_R' = $SO_R \cup \{(w,v)/\{w \mapsto read(r)\} \vee \{w \mapsto update(u)_j\}\}$. Applying the intersection $(\mathbb{U}\times\mathbb{R})$, we shall obtain $SO_R \cup \{(w,v)/\{w \mapsto update(u)_j\}\}$.
  - VIS$' =$ VIS $\cup \{(x,v)/\{x \mapsto update(u)_h\} \in$ OP $\wedge\ u^x \in S[0..k-1]\cdot\rho\cdot[\alpha]\}$.

  By inductive hypothesis, $SO_R \cup (\mathbb{U}\times\mathbb{R}) \subseteq$ VIS. We only have to prove that $\{(w,v)/\{w \mapsto update(u)_j\} \in \text{OP}\} \subseteq \{(x,v)/\{x \mapsto update(u)_h\} \in \text{OP} \wedge u^x \in S[0..k-1]\cdot\rho\cdot[\alpha]\}$. When $j = h$ and $w = x$, we can use Lemma 3.1. So that, we have proved that SO_R' $\cap (\mathbb{U}\times\mathbb{R}) \subseteq$ VIS'.

Theorem 3.2 (MONOTONIC READ). *Let* SO_R *the second component of the relation* SO *and* VIS *a visibility relation, if* $\{N_0,\emptyset,\emptyset,\emptyset,\emptyset\} \to^* \{N, \text{OP}, \text{SO}, \text{VIS}, \text{AR}\}$ *then* $(\text{VIS}; SO_R) \cap (\mathbb{U}\times\mathbb{R}) \subseteq$ VIS

Proof. The proof follows by induction on the length of the derivation $\to^*$.

- **n=0**. In particular SO_R and VIS are $\emptyset$, so that $\emptyset \subseteq \emptyset$.
- **n=k+1**. Then $\{N_0,\emptyset,\emptyset,\emptyset,\emptyset\} \to^n \{N, \text{OP}, \text{SO}, \text{VIS}, \text{AR}\} \xrightarrow{\alpha}_i \{N', \text{OP}', \text{SO}', \text{VIS}', \text{AR}\}$. Let R be a composition of relations. We shall say that if $(x,y) \in R$ iff $\exists y \in \mathbb{V}$ such that $(x,y) \in \text{VIS}' \wedge (y,z) \in SO_R$'. We have to prove that $(x,z) \in$ VIS'. We proceed by case analysis on the last transition:

- **rule (E-READ)**. We know that v is fresh, therefore, v have not be neither x nor y because there exists z such that, z happens after from x and y. There are only two possibilities:
  - $z = v$. As $(y,z) \in \ \mathrm{SO}_R$, then y and z are from the same client, called i. We are only interested in relations of $update \times \ read$. We know that v is associated to an read action besides x have to be an update action. As $(x,y) \in$ VIS' then y is an read action,i.e., $\{y \mapsto read(r)_i\} \in$ OP.By Lemma xx, there exists an update u such that $u^x \in \ S[0..k_i-1] \cdot \rho_i \cdot [\alpha_i]$. Substituting w by v in VIS', we prove that $(x,z) \in$ VIS'. item $z \neq v$. This case follows immediately by inductive hypothesis.
- **rule (E-UPDATE)**. Visibility relation does not change,i.e.,VIS = VIS'. Let v be a vertice associated an update action, then SO_R' = $\mathrm{SO}_R \cup \ \{(w,v)/\{w \mapsto read(r)\} \vee \ \{w \mapsto \ update(u)_j\}\}$. As we are only interested in relations of $update \times read$, then $(\mathrm{VIS}'; \mathrm{SO'}_R) \cap (\mathbb{U} \times \mathbb{R}) \equiv (\mathrm{VIS}; \mathrm{SO}_R) \cap (\mathbb{U} \times \mathbb{R})$. By inductive hypothesis we can prove that $(\mathrm{VIS}'; \mathrm{SO}_R)' \cap (\mathbb{U} \times \mathbb{R}) \subseteq \mathrm{VIS} \subseteq$ VIS'. The proof for the remaining cases follow are not interesting because the relations does not change.

Theorem 3.3 (NO CIRCULAR CAUSALITY).
Let SO_R *the second component of the relation* SO *and* VIS *a visibility relation, if* $\{N_0, \emptyset, \emptyset, \emptyset, \emptyset\} \rightarrow^* \ \{N, \mathrm{OP}, \mathrm{SO}, \mathrm{VIS}, \mathrm{AR}\}$ *then* ***acyclic*** $(\mathrm{SO}_R \cup \mathrm{VIS})^+$.

Proof. Since VIS is acyclic and $\mathrm{SO}_R \cap (\mathbb{U} \times \mathbb{R}) \subseteq$ VIS by Theorem 3.1, we have to prove that VIS^+ is acyclic. In particular, the transitive closure of an acyclic graph is the reachability relation of the directed acyclic graph and a strict partial order.

Theorem 3.4 (CAUSAL VISIBILITY).
Let SO_R *the second component of the relation* SO *and* VIS *a visibility relation,* $(\mathrm{SO}_R \cup \mathrm{VIS})^+ \cap (\mathbb{U} \times \mathbb{R}) \subseteq \mathrm{VIS}$.

Proof. The proof follows by induction on the number of union sets between SO_R and VIS. Then, $\bigcup_{n=1}^{\infty} (\mathrm{SO}_R \cup \mathrm{VIS})^n \cap (\mathbb{U} \times \mathbb{R}) \subseteq \mathrm{VIS}$.

- **n=0**. This means that SO_R and VIS are $\emptyset$, so that $\emptyset \subseteq \emptyset$.
- **n=k+1**. Suppose that we have proved that the number of union sets $< k+1$. Now, we have to prove that: $\forall (a,b) \mid (a,b \in \mathbb{V} \Rightarrow \ (a,b) \ \in \ (\mathrm{SO}_R \cup \mathrm{VIS})^{k+1} \cap (\mathbb{U} \times \mathbb{R})) \Rightarrow \ (a,b) \in \mathrm{VIS}$.
  Assume $(a,x_1),(x_1,x_2),\ldots(x_{k-1},x_k),(x_k,b)$ are relations from $(\mathrm{SO}_R \cup \mathrm{VIS})^{k+1}$. Then $(a,x_1),(x_1,x_2),\ldots(x_{k-1},x_k)$ are relations from $(\mathrm{SO}_R \cup \mathrm{VIS})^k$. By the induction hypothesis, $(a,x_k) \in (\mathrm{SO}_R \cup \mathrm{VIS})^k$, and we also have $(x_k,b) \in (\mathrm{SO}_R \cup \mathrm{VIS})$. Thus by the definition of $(\mathrm{SO}_R \cup \mathrm{VIS})^{k+1}$, $(a,b) \in (\mathrm{SO}_R \cup \mathrm{VIS})^{k+1}$. Conversely, assume $(a,b) \in \ (\mathrm{SO}_R \cup \mathrm{VIS})^{k+1} = (\mathrm{SO}_R \cup \mathrm{VIS})^k \circ (\mathrm{SO}_R \cup \mathrm{VIS})$. Then there is a vertice $c \in \mathbb{V}$ such that $(a,c) \in \ (\mathrm{SO}_R \cup \mathrm{VIS})^k$ and $(c,b) \in \ (\mathrm{SO}_R \cup \mathrm{VIS})$.
  We are only interested when a is an update action and b a read action. It is because of the intersection with $(Update \times Read)$. We have two possible cases:
  - c is a read action. It means that (c,b) only can be in SO_R because VIS requires that c will be an update action. In particular, if $(c,b) \in \ \mathrm{SO}_R$, they belong to the same client.

  - if $(a,c) \in$ VIS and $(c,b) \in \text{SO}_R$, by Theorem 3.2, $(a,b) \in$ VIS.
  - if $(a,c) \in \text{SO}_R$ then by Theorem 3.1, $(a,c) \in$ VIS- Then, it is analogous to the previous case.
- c is an update action. It means that (a,c) only can be in SO_R because VIS requires that c will be an read action. In particular, if $(a,c) \in \text{SO}_R$, they belong to the same client.
  - if $(a,c) \in \text{SO}_R$ and $(c,b) \in$ VIS, by Theorem Monotonic Writes (Falta probar!), $(a,b) \in$ VIS.
  - if $(a,c) \in$ VIS ... VER.

Theorem 3.5 (CAUSAL ARBITRATION).
Let SO_R *the second component of the relation* SO*,* VIS *a visibility relation and* AR *an arbitration relation then* $(\mathbb{U} \times \mathbb{U}) \cap (\text{SO}_R \cup \text{VIS})^+ - \text{SO}_R \subseteq \text{AR}$.

Proof. The proof follows by induction on the number of union sets between SO_R and VIS. Then, $(\mathbb{U} \times \mathbb{U}) \cap \bigcup_{n=1}^{\infty} (\text{SO}_R \cup \text{VIS})^n - \text{SO}_R \subseteq \text{AR}$.

- **n=0**. This means that SO_R, VIS and AR are $\emptyset$, so that $\emptyset \subseteq \emptyset$.
- **n=k+1**. Suppose that we have proved that the number of union sets $< k+1$. Now, we have to prove that: $\forall (a,b) \mid (a,b \in \mathbb{V} \Rightarrow (a,b) \in (\mathbb{U} \times \mathbb{U}) \cap \bigcup_{n=1}^{\infty} (\text{SO}_R \cup \text{VIS})^{k+1} - \text{SO}_R \Rightarrow (a,b) \in \text{AR}$.
  Assume $(a,b) \in (\text{SO}_R \cup \text{VIS})^{k+1} = (\text{SO}_R \cup \text{VIS})^k \circ (\text{SO}_R \cup \text{VIS})$. Then there is a vertice $c \in \mathbb{V}$ such that $(a,c) \in (\text{SO}_R \cup \text{VIS})^k$ and $(c,b) \in (\text{SO}_R \cup \text{VIS})$.
  We are only interested when a and b are an update actions. It is because of the intersection with $(Update \times Update)$. We have two possible cases:
  - c is a read action. It means that (c,b) only can be in SO_R because VIS requires that c will be an update action. In particular, if $(c,b) \in \text{SO}_R$, they belong to the same client. COMPLETAR
  - c is an update action. COMPLETAR

Example 3.1 (Consistent Prefix). Consider the following system built-up from two different client:

$$E = \{\{update(aList.add('a'));push();,0,\varepsilon,\varepsilon,\varepsilon,0\}_1 \parallel \{update(aList.add('b'));push();let\ v = read(x);,0,\varepsilon,\varepsilon,\varepsilon,0\}_2 \parallel \varepsilon,\emptyset,\emptyset,\emptyset,\emptyset\}$$

Environment has a system N with two clients and a message queue S without updates. Relation sets OP,SO,VIS,AR are empty, i.e.,there were not an execution in N captured by the relation sets. The update action add a string to an object called *aList* which has not elements. In this state, C_1 and C_2 can perform their update actions. E may nondeterministically choose to do $update(aList.add('a')$ or $update(aList.add('b')$. If the first communication takes place over $update(aList.add('a')$, then the system evolves as follows:

$$E \xrightarrow{update(aList.add('a')^{v_0})}_1 \{\{push();,0,\varepsilon,[aList.add('a')],\varepsilon,0\}_1 \parallel \{update(aList.add('b'));push();let\ x = read(aList);,0,\varepsilon,\varepsilon,\varepsilon,0\}_2 \parallel \varepsilon, \emptyset,\{(\{v_0\},\emptyset)\},\emptyset,\emptyset\}$$

The action $update(aList.add('a'))$ of the client 1 will be identified by vertice v_0 by rule E-UPDATE. In particular, the update action will be left into the transactional queue of Client 1 besides the new vertice will be added to vertices in SO. Now, Client 1 performs *push()*:

$$E \xrightarrow{push()}_1 \{\{0,0,[aList.add('a')],\varepsilon,[aList.add('a')],0\}_1 \parallel \{update(aList.add('b'));push();let\ x = read(aList);,0,\varepsilon,\varepsilon,\varepsilon,0\}_2 \parallel \varepsilon, \emptyset,\{(\{v_0\},\emptyset)\},\emptyset,\emptyset\}$$

When it happens, $aList.add('a')$ is moved to the sent queue and pending queue. At this moment, Client 2 can perform an update action however Client 1 will realize an internal action which is given by E-PROCESS.

$$E \xrightarrow{\tau}_1 \{\{0,0,[aList.add('a')],\varepsilon,\varepsilon,0\}_1 \parallel \{update(aList.add('b'));push();let\ x = read(aList);,0,\varepsilon,\varepsilon,\varepsilon,0\}_2 \parallel aList.add('a'),\emptyset,\{(\{v_0\},\emptyset)\},\emptyset,\emptyset\}$$

Analogously, Client 2 realize its actions leaving to environment evolves as below:

$$E \xrightarrow{update(aList.add('b'))}_2 \{\{0,0,[aList.add('a')],\varepsilon,\varepsilon,0\}_1 \parallel \{push();let\ x = read(aList);,0,\varepsilon,[aList.add('b')],\varepsilon,0\}_2 \parallel aList.add('a'),\emptyset,\{(\{v_0,v_1\},\{(v_0,v_1)\})\},\emptyset,\emptyset\}$$

$$\xrightarrow{push()}_2 \{\{0,0,[aList.add('a')],\varepsilon,\varepsilon,0\}_1 \parallel \{let\ x = read(aList);,0,[aList.add('b')],\varepsilon,[aList.add('b')],0\}_2 \parallel aList.add('a'),\emptyset,\{(\{v_0,v_1\},\{(v_0,v_1)\})\},\emptyset,\emptyset\}$$

$$\xrightarrow{\tau}_2 \{\{0,0,[aList.add('a')],\varepsilon,\varepsilon,0\}_1 \parallel \{let\ x = read(aList);,0,[aList.add('b')],\varepsilon,\varepsilon,0\}_2 \parallel aList.add('a')\cdot[aList.add('b')],\emptyset,\{(\{v_0,v_1\},\{(v_0,v_1)\})\},\emptyset,\{(v_0,v_1)\}\}$$

$$\xrightarrow{read(aList)}_2 \{\{0,0,[aList.add('a')],\varepsilon,\varepsilon,0\}_1 \parallel \{0[x\mapsto['b']],0,[aList.add('b')],\varepsilon,\varepsilon,0\}_2 \parallel aList.add('a')\cdot[aList.add('b')], ,\emptyset,\emptyset,\emptyset\}\emptyset$$

Note that AR is modified because the message queue has two elements. The value returned will be the list with a only element, ['b'], because of the internal action associated to the rule E-RECEIVE did not perform it. Then, if the rule E-RECEIVE were performed, the value of the list would be ['a','b']. It is because the content of the pending queue is removed when the server left their message.

The guarantee Consistent Prefix, which rules is $(\text{AR};\text{VIS}) \subseteq \text{AR};\text{VIS}$, states that if we see an result from a client in a particular order, we will never see this result in a different order.

4 Extending GSP

As illustrated in above examples, there are situations in which a environment does not ensures consistency guarantees as Consistent Prefix or Single Order. The nature of such cases is caused by having no transactions as the usual databases.

The remaining of this section is proposing a environment that ensures these consistency guarantees. We start by refining the environment in GSP. We will distinguish two relations:

Return Before relates updates actions. It indicates the ordering of non-overlapping operations.

Transactions relates vertices from a client to set of naturals.

$$\begin{array}{ll}(\text{ENVIRONMENT}) & E ::= \{N, \text{OP}, \text{SO}, \text{VIS}, \text{AR}, \text{RB}, \text{Tx}\} \\ (\text{TRANSACTIONS}) & \text{Tx} ::= \emptyset \mid v^i \mapsto \mathcal{N}, \text{Tx}\end{array}$$

We assume the following countable sets of transactional vertices names $[\mathbb{V}]$ ranged over by $[v]; [v_0]; [v_1], \ldots;$

The labeled transition system for the extension of GSP considers the following actions β:

$$\beta ::= \alpha \mid \textit{start-sync} \mid \textit{end-sync} \mid [\alpha]$$

These labels allow system to perform an update synchronous. Label *start-sync* stands for the begining of a transaction, and *end-sync* the end of it.

ENVIRONMENT

(E-START-SYNC)

$$\frac{N \xrightarrow{\textit{sync\_update}(u^v)}_i N' \qquad [v^i] \notin \textit{dom}(\text{Tx})}{\{N, \text{OP}, \text{SO}, \text{VIS}, \text{AR}, \text{RB}, \text{Tx}\} \xrightarrow{\textit{start-sync}}_i \{N', \text{OP}, \text{SO}, \text{VIS}, \text{AR}, \text{RB}, \text{Tx}\}}$$

(E-END-SYNC)

$$\frac{N \xrightarrow{\textit{end-sync}(u^{[v]})}_i N' \qquad [v^i] \in \textit{dom}(\text{Tx})}{\{N, \text{OP}, \text{SO}, \text{VIS}, \text{AR}, \text{RB}, v^i \mapsto \mathcal{N}, \text{Tx}\} \xrightarrow{\textit{end-sync}([v^i])}_i \{N', \text{OP}, \text{SO}, \text{VIS}, \text{AR}, \text{RB}', \text{Tx}\}}$$

PROGRAM

(T-SYNC-UPDATE)

$$\{\mathit{sync\_update(u)};P,k,\rho,\alpha,\sigma,j\}_i \parallel C \parallel S \xrightarrow{\mathit{sync\_update}(u^v)}_i \{[\mathit{update(u)};\mathit{flush()}];P,k,\rho,\alpha,\sigma,j\}_i \parallel C \parallel S$$

(T-TRAN)

$$\frac{\{P,k,\rho,\alpha,\sigma,j\}_i \parallel C \parallel S \xrightarrow{\alpha}_i \{P',k',\rho',\alpha',\sigma',j'\}_i \parallel C \parallel S}{\{[P];Q,k,\rho,\alpha,\sigma,j\}_i \parallel C \parallel S \xrightarrow{[\alpha]}_i \{[P'];Q,k',\rho',\alpha',\sigma',j'\}_i \parallel C \parallel S}$$

(T-FLUSH)

$$\{[\mathit{flush()}];P,k,\rho,\alpha,\sigma,j\}_i \parallel C \parallel S \xrightarrow{\tau}_i \{[\mathit{while}(\mathit{cond})\ \mathbf{do}\ Q];P,k,\rho,\alpha,\sigma,j\}_i \parallel C \parallel S$$

(T-END-SYNC)

$$\{[0];P,k,\rho,\alpha,\sigma,j\}_i \parallel C \parallel S \xrightarrow{\mathit{end\text{-}sync}(u^{[v]})}_i \{P,k,\rho,\alpha,\sigma,j\}_i \parallel C \parallel S$$