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 vertex 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:*

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

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 ith 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:

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

(T-READ)

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

(T-UPDATE)

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

(T-PUSH)

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

(T-PULL)

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

(T-CONFIRMED)

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

(T-WHILE-TRUE)

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

(T-WHILE-FALSE)

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

(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 *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   Implementation of GSP

The GSP model introduced in the previous section is an abstract protocol which does not consider implementation details. Problems like space explosion caused by the growing of the update sequence have to allow for implementing these protocol.

Instead of storing update sequence, the implementation to present reduce such sequence to in two differents objects. *State* when the update sequence reduced represents a prefix of the global queue message or *Delta* when the update sequence reduced represents a interval or segment of the global queue message.

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

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

$$
\begin{array}{lll}
\textbf{const} & \textit{initialstate} & :State \\
\textbf{function } \textit{read} & : Read \times State \to Value \\
\textbf{function } \textit{apply} & : State \times Delta^* \to State \\
\textbf{const} & \textit{emptydelta} & : Delta \\
\textbf{function } \textit{append} & : Delta \times Update \to Delta \\
\textbf{function } \textit{reduce} & : Delta^* \to Delta
\end{array}
$$

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:

$\rho, \sigma := \varepsilon \mid [\delta] \cdot \rho$

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

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

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

2.1 Auxiliar Function

Before describing the implementation of GSP, we will introduce the following auxiliars functions which will be used when we define the operational semantic.

```
append :: GSSegment x Round* → State
append(<δ, mr>, ε) = <δ, mr>
append(<δ, mr>, <i,n₀,δ₀>:rs) = append(<reduce(δ · δ₀ · ε), mr[b₀ ↦ n₀]>,
rs)

apply :: GSPrefix x GSSegment → GSPrefix
apply(<ps,mr>, <δ, mr'>) = <apply(ps, δ · ε), mr[mr']>

receivedrounds :: (N → <N × Delta>*) → Round*
receivedrounds(⊥) = ε
receivedrounds(b → <n₀, δ₀>· f) = <b, n₀, δ₀> · receivedrounds(f)

curstate :: State x Round* x Delta x Delta → State
curstate(ps, p, pb, tb) = apply(ps, getdeltas(p) · pb· tb)

getdeltas :: Round* → Delta*
getdeltas(ε) = ε
getdeltas(<n₀,δ₀ >· δ) = δ₀ · getdeltas(δ)
```

The functions **reduce** and **apply** are abstract and depend on the data model used.

2.2 Sintax

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

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

The implementation of a system is a server and clients interacting concurrently. A server will be represented with a persisted state denoted by *ps*, input messages in_s and output messages out_s. We will refer to clients, as tuple of program *P* and set of states and sequences *E*. A program client stores a state ($E.k$), a pending queue ($E.p$),i.e., updates sent to the server without confirmation of their reception, a push buffer ($E.pb$) which holds updates that were pushed by the client but have not been sent to the server, a transaction buffer ($E.tb$) which holds updates for sending with another ones, a receive buffer ($E.GSSegment \cup GSPrefix$) for updates which were sent by the server, the number of round sent ($E.n$), a input message queue ($E.in_c$) and a output message queue ($E.out_c$).

2.3 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)
$$\{ps, in_s, out_s\} \xmapsto{\ddagger} \{ps, \bot, \bot\}$$

(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)

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

(UPDATE)

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

(PUSH)

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

(PULL)

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

(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 Equivalence

We have introduced an abstract GSP protocol and a robust streaming server-client implementation of GSP. There exists a representation relation $\triangleleft$ that relates state and delta objects to the update sequences.

– On $Delta \times Update^*$, let $\triangleleft$ be the smallest relation such that (1) $emptydelta \triangleleft []$, and (2) $d \triangleleft a$ implies $append(d,u) \triangleleft a \cdot u$ for all updates u, and (3) $d_1 \triangleleft a_1 \ldots d_n \triangleleft a_n$ implies $reduce(d_1 \ldots d_n) \triangleleft a_1 \cdots a_n$.

– On $State \times Update^*$, let $\triangleleft$ be the smallest relation such that (1) $initialstate \triangleleft []$, and (2) $s \triangleleft a \wedge d_1 \triangleleft a_1 \wedge \ldots \wedge d_n \triangleleft a_n$ implies $apply(s, d_1 \ldots d_n) \triangleleft a \cdot a_1 \cdots a_n$.

Next theorem states one of most important result of the paper, saying that they are weak bisimulation equivalent.

Theorem 3.1. *Let* $\mathcal{R}$ *be a relation defined as* $\{(\|_{i \in I_{\{0 \ldots n\}}} A_i \parallel S_A), (\|_{i \in I_{\{0 \ldots n\}}} C_i \parallel S_C) \mid \forall i \in \mathbb{N}, \forall r : Read, A_i = \{P_i, k_i, \rho_i, \alpha_i, \sigma_i, j_i\} \wedge C_i = \{P_i, E_i\} \wedge rvalue(r, flatten(S_A[0..k-1] \cdot \rho) \cdot \alpha) = \textbf{read}(r, curstate(E_i.k, E_i.p, E_i.pb, E_i.tb)) \wedge S_C.ps \triangleleft S_A \wedge E_i.tb = [\alpha_i] \wedge E_i.rb[0].delta \triangleleft S_A[k_i] \ldots E_i.rb[j-1].delta \triangleleft S_A[k_i + j_i] - 1 \wedge E_i.known \triangleleft S_A[0 \ldots k_i - 1] \wedge E_i.p \cdot E_i.pb \triangleleft \rho_i \wedge \forall l, 1 \leq l < |in_c|, E_i.in_c[l] \triangleleft S_A[k+j-1+l] \wedge k+j+1 \leq |S_A| \Leftrightarrow |E_i.in_c| + |S_C.out_s| > 0 \wedge \forall \beta \in \sigma_i, \beta \in E_i.out_c \cup S_C.in_s\}.$*

Proof. We have to prove that $\mathcal{R}$ satisfies the nine points of its definition:

1. $rvalue(r, flatten(S_A[0..k-1] \cdot \rho) \cdot \alpha) = \textbf{read}(r, curstate(E_i.k, E_i.p, E_i.pb, E_i.tb))$
2. $S_C.ps \triangleleft S_A$
3. $E_i.tb = [\alpha_i]$
4. $E_i.rb[0].delta \triangleleft S_A[k_i] \ldots E_i.rb[j-1].delta \triangleleft S_A[k_i + j_i] - 1$
5. $E_i.known \triangleleft S_A[0 \ldots k_i - 1]$
6. $E_i.p \cdot E_i.pb \triangleleft \rho_i$
7. $\forall l, 1 \leq l < |in_c|, E_i.in_c[l] \triangleleft S_A[k+j-1+l]$
8. $k+j+1 \leq |S_A| \Leftrightarrow |E_i.in_c| + |S_C.out_s| > 0$
9. $\forall \beta \in \sigma_i, \beta \in E_i.out_c \cup S_C.in_s$

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

– **n=0**.
– **n=k+1**. $\forall (A,C) \in \mathcal{R}$
  • **rule (T-READ)**. Then $A = \{let\ x = read(r); P, k, \rho, \alpha, \sigma, j\}_i \parallel \tilde{A} \parallel S_A$ and $A' = \{P[x \mapsto v], k, \rho, \alpha, \sigma, j\}_i \parallel \tilde{A} \parallel S_A$. As (A,C) belongs $\mathcal{R}$ then C must be $\{\textbf{read}(r, curstate(k, p, pb, tb))\}_i \parallel \tilde{C} \parallel S_C$, so that, read transition is the only one that C could perform. Then, $C' = \{P[x \mapsto v], E\}_i \parallel \tilde{C} \parallel S_C$. If $(A',C') \in \mathcal{R}$ then we will check the previous nine properties:
    1. Note that $v = v'$

4 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 vertex v, we shall write $\text{SO} \triangleright_i v$ meaning that $(\mathcal{V}, \mathcal{R}) \triangleright_i v = (\mathcal{V} \cup \{v\}, \mathcal{R} \cup \{(x,v)/x \in \mathbb{V}\})$. 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)

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

(E-UPDATE)

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

4.1 Ordering Guarantees

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

First, we prove a useful lemma:

Lemma 4.1. *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 $\{N, \emptyset, \emptyset, \emptyset, \emptyset\} \to^n \{N, \text{OP}, \text{SO}, \text{VIS}, \text{AR}\} \to_i \{x \mapsto update(u)_i\} \in \text{OP} \Rightarrow$
$u^x \in S[0..k_i - 1] \cdot \rho_i \cdot [\alpha_i]$. 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 $\{x \mapsto update(u)_i\} \in \text{OP}$, 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 $N \xrightarrow{push()}_i N'$, $k_i' = k_i$, $\rho_i' = \rho_i \cdot [\alpha_i]$ and $\alpha_i' = \varepsilon$.
- **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 4.1 (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 $\text{SO}_R \cap (\mathbb{U} \times \mathbb{R}) \subseteq \text{VIS}$

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:
  - $\text{SO}_R' = \text{SO}_R \cup \{(w,v)/\{w \mapsto read(r)\} \vee \{w \mapsto update(u)_j\}\}$. Applying the intersection $(\mathbb{U} \times \mathbb{R})$, we shall obtain $\text{SO}_R \cup \{(w,v)/\{w \mapsto update(u)_j\}\}$.
  - $\text{VIS}' = \text{VIS} \cup \{(x,v)/\{x \mapsto update(u)_h\} \in \text{OP} \wedge u^x \in S[0..k - 1] \cdot \rho \cdot [\alpha]\}$.

  By inductive hypothesis, $\text{SO}_R \cup (\mathbb{U} \times \mathbb{R}) \subseteq \text{VIS}$. 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 4.1. So that, we have proved that $\text{SO}_R' \cap (\mathbb{U} \times \mathbb{R}) \subseteq \text{VIS}'$.

Theorem 4.2 (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 $(\text{VIS}; \text{SO}_R) \cap (\mathbb{U} \times \mathbb{R}) \subseteq \text{VIS}$*

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 $\{N_0, \emptyset, \emptyset, \emptyset, \emptyset\} \rightarrow^n \{N, \text{OP}, \text{SO}, \text{VIS}, \text{AR}\} \xrightarrow{\alpha}_i \{N', \text{OP}', \text{SO}', \text{VIS}', \text{AR}\}$. Let R be a composition of relations. We shall say that if $(x, y) \in R$ iff $\exists y \in \mathbb{V}$ such that $(x, y) \in \text{VIS}' \wedge (y, z) \in \text{SO}_R$'. We have to prove that $(x, z) \in \text{VIS}'$. 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 $(y, z) \in \text{SO}_R$, 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 $(x, y) \in \text{VIS}'$ then y is an read action, i.e., $\{y \mapsto read(r)_i\} \in \text{OP}$. 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 $(x, z) \in \text{VIS}'$. 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 vertex associated an update action, then $\text{SO}_R' = \text{SO}_R \cup \{(w, v)/\{w \mapsto read(r)\} \vee \{w \mapsto update(u)_j\}\}$. As we are only interested in relations of $update \times read$, then $(\text{VIS}'; \text{SO}'_R) \cap (\mathbb{U} \times \mathbb{R}) \equiv (\text{VIS}; \text{SO}_R) \cap (\mathbb{U} \times \mathbb{R})$. By inductive hypothesis we can prove that $(\text{VIS}'; \text{SO}_R)' \cap (\mathbb{U} \times \mathbb{R}) \subseteq \text{VIS} \subseteq \text{VIS}'$. The proof for the remaining cases follow are not interesting because the relations does not change.

Theorem 4.3 (NO CIRCULAR CAUSALITY).

Let SO_R *the second component of the relation* SO *and* VIS *a visibility relation, if* $\{N_0, \emptyset, \emptyset, \emptyset, \emptyset\} \rightarrow^* \{N, \text{OP}, \text{SO}, \text{VIS}, \text{AR}\}$ *then* ***acyclic***$(\text{SO}_R \cup \text{VIS})^+$.

Proof. Since VIS is acyclic and $\text{SO}_R \cap (\mathbb{U} \times \mathbb{R}) \subseteq \text{VIS}$ by Theorem 4.1, 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 4.4 (CAUSAL VISIBILITY).

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

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

- **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: $\forall (a, b) \mid (a, b \in \mathbb{V} \Rightarrow (a, b) \in (\text{SO}_R \cup \text{VIS})^{k+1} \cap (\mathbb{U} \times \mathbb{R})) \Rightarrow (a, b) \in \text{VIS}$.
  Assume $(a, x_1), (x_1, x_2), \ldots (x_{k-1}, x_k), (x_k, b)$ are relations from $(\text{SO}_R \cup \text{VIS})^{k+1}$. Then $(a, x_1), (x_1, x_2), \ldots (x_{k-1}, x_k)$ are relations from $(\text{SO}_R \cup \text{VIS})^k$. By the induction hypothesis, $(a, x_k) \in (\text{SO}_R \cup \text{VIS})^k$, and we also have $(x_k, b) \in (\text{SO}_R \cup \text{VIS})$. Thus by the definition of $(\text{SO}_R \cup \text{VIS})^{k+1}$, $(a, b) \in (\text{SO}_R \cup \text{VIS})^{k+1}$. Conversely, 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 vertex $c \in \mathbb{V}$ 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 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 $(c,b) \in \text{SO}_R$, they belong to the same client.
    * if $(a,c) \in \text{VIS}$ and $(c,b) \in \text{SO}_R$, by Theorem 4.2, $(a,b) \in \text{VIS}$.
    * if $(a,c) \in \text{SO}_R$ then by Theorem 4.1, $(a,c) \in \text{VIS}$- 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 $(c,b) \in \text{VIS}$, by Theorem Monotonic Writes (Falta probar!), $(a,b) \in \text{VIS}$.
    * if $(a,c) \in \text{VIS}$ ... VER.

Theorem 4.5 (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 vertex $c \in \mathbb{V}$ 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 4.1 (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 non-deterministically 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:

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

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

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

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.

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

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

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

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

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

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

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.

5   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 a new relation:

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

$$
\begin{array}{ll}
(\textsc{environment}) & E ::= \{N, \textsc{op}, \textsc{so}, \textsc{vis}, \textsc{ar}, \textsc{rb}, \textsc{to}, \textsc{tc}\} \\
(\textsc{transactions with overlapping}) & \textsc{to} ::= \emptyset \mid w^{j} \mapsto \mathbb{N}, \textsc{to} \; with \; w \, \in \, \mathbb{V} \, j \in \, \mathbb{N} \\
(\textsc{transactions closed}) & \textsc{tc} ::= \varepsilon \mid w^{j} \cdot \textsc{tc} \; with \; w \, \in \, \mathbb{V} \, j \in \, \mathbb{N}
\end{array}
$$

The term $\textsc{to}$ relates vertex from a client to set of vertex which have just not finished; $\textsc{tc}$ denotes transaction closed.

5.1   Operational Semantic

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:

$$
\beta ::= \alpha \mid \dagger \mid \ddagger \mid [\alpha]
$$

These labels allow system to perform an update synchronous. Label $\dagger$ stands for the begining of a transaction, and $\ddagger$ the end of it.

ENVIRONMENT

(E-START-SYNC)

$$\frac{N \xrightarrow{sync\_update(u^v)}_i N' \qquad [v^i] \notin dom(\text{TO})}{\{N, \text{OP}, \text{SO}, \text{VIS}, \text{AR}, \text{RB}, \text{TO}, \text{TC}\} \xrightarrow{\dagger}_i \{N', \text{OP}, \text{SO}, \text{VIS}, \text{AR}, \text{RB}, \text{TO}; v^i \mapsto dom(\text{TC}), \text{TC}\}}$$

(E-END-SYNC)

$$\frac{N \xrightarrow{\ddagger(u^{[v]})}_i N' \qquad [v^i] \in dom(\text{TO})}{\{N, \text{OP}, \text{SO}, \text{VIS}, \text{AR}, \text{RB}, v^i \mapsto \mathcal{N}, \text{TO}, \text{TC}\} \xrightarrow{\ddagger([v^i])}_i \{N', \text{OP}, \text{SO}, \text{VIS}, \text{AR}, \text{RB}', \text{TO}, \text{TC} \cdot v^i\}}$$

where $\text{RB}' = \text{RB} \cup \{(x,v) \mid \forall 0 \leq m,n < |S|, x \mapsto S[m] \wedge v \mapsto S[n] \wedge m \leq n \wedge x \notin \text{TC}(v)\}$

PROGRAM

(T-SYNC-UPDATE)

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

(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)

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

(T-END-SYNC)

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