

# GSP-Calculus

Hernán Melgratti<sup>1,2</sup> and Christian Roldán<sup>1</sup>

<sup>1</sup> Departamento de Computación, FCEyN, Universidad de Buenos Aires.

<sup>2</sup> CONICET.

**Abstract.**

## 1 Abstract Global Sequence Protocol Language

### 1.1 Syntax

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

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

Let  $\rho$  and  $\alpha$  be sequence, we write  $\setminus$  as relative complement of  $\rho$  in  $\alpha$ , 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:*

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

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  $i^{\text{th}}$  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  $\rho$ , a transaction buffer  $\alpha$ , a sequence of all the updates sent by the client to the RTOB  $\sigma$  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)

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

(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)

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

(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)

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

(T-WHILE-TRUE)

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

(T-WHILE-FALSE)

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

(T-RECEIVE)

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

(T-PROCESS)

$$\{P, k, \rho, \alpha, [\beta] \cdot \sigma, j\}_i \parallel C \parallel S \xrightarrow{\tau} \{P, k, \rho, \alpha, \sigma, j\}_i \parallel C \parallel S \cdot \beta$$

The meaning of  $\tau$  is standard. A *tgsp-calculus* program is given by a Labeled Transition System (LTS) before introducing. Transitions for systems are labeled by actions  $\alpha$  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  $\alpha$ . After (T-PUSH) the content of the transaction queue  $\alpha$  is moved to the pending queue,  $\rho$ , and the sent queue,  $\sigma$ . 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

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

We write  $[x_1 \mapsto y_1, \dots, x_n \mapsto y_n]$  for the partial function  $f$  such that  $dom(f) = \{x_1, \dots, x_n\}$  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 different 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  $State \times \mathbb{N} \rightarrow \mathbb{N}$ , *GSSegment* the set operation over sets  $Delta \times \mathbb{N} \rightarrow \mathbb{N}$  and *Round* the set over sets  $\mathbb{N} \times \mathbb{N} \times Delta$ ;  $\theta$  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  $\rightarrow$  Value
function apply       : State  $\times$  Delta*  $\rightarrow$  State
const    emptydelta : Delta
function append      : Delta  $\times$  Update  $\rightarrow$  Delta
function reduce      : Delta*  $\rightarrow$  Delta

```

Let *State* be a set of state ranged over by  $s, s_1, s_2, \dots$ ; We shall use  $\delta$  to denote sequence of *Delta* ranged over by  $d, d_0, d_1 \dots$ ; Let  $\rho, \sigma$  be sequence defined:

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

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

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

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

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

apply :: GSPrefix × 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 × Round* × Delta × 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)

remove :: Delta* → Delta*
remove(rs, ε) = ε
remove(rs, xs) = (foldr(\x rec ys → h (filter (/=x)ys)) id)xs rs

```

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

## 2.2 Syntax

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

```

(SYSTEM)   N ::= 0 ∣ S ∥ C
(SERVER)   S ::= 0 ∣ {ps, ins, outs}
(CLIENTS)  C ::= ∅ ∣ {P, <State, ρ, δ, GSSegment ∪ GSPrefix, N, inc, outc>} ∣ C ∥ C
(PROGRAM)  P ::= 0 ∣ let x = read(r); P ∣ update(u); P ∣ push(); P ∣ 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

$$\begin{array}{c} \text{(DROP\_CONN)} \\ \frac{b_i \in in_s \quad b_i \in out_s}{\{ps, in_s, out_s\} \xrightarrow{\dagger(b_i)} \{ps, in_s \setminus b_i, out_s \setminus b_i\}} \quad \text{(CRASH\_AND\_RECOVER)} \\ \{ps, in_s, out_s\} \xrightarrow{\ddagger} \{ps, \perp, \perp\} \\ \\ \text{(BATCH)} \\ \frac{rs = receivedRounds(in_s) \quad gs = append(\theta, rs) \quad ps' = \mathbf{apply}(ps, gs.delta)}{\{ps, in_s, out_s\} \xrightarrow{\tau} \{ps', in_s, notify(dom(out_s), out_s, gs)\}} \\ \\ \text{(ACCEPT\_CONN)} \\ \frac{b_i \notin in_s \quad b_i \notin out_s}{\{ps, in_s, out_s\} \xrightarrow{\oplus(b_i)} \{ps, in_s, out_s[b_i \mapsto ps]\}} \end{array}$$

#### COMMUNICATION

$$\begin{array}{c} \text{(COMM SERVER-CLIENT)} \\ \frac{out_s(i) = gs \cdot gss}{\{ps, in_s, out_s\} \parallel \{P, E\}_i \xrightarrow{\tau} \{ps, in_s, out_s[i \mapsto gss]\} \parallel \{P, E[in_c \mapsto E.in_c \cup \{gs\}]\}_i} \\ \\ \text{(COMM CLIENT-SERVER)} \\ \frac{out_c = \langle i, n_0, \delta_0 \rangle \cdot rs}{\{ps, in_s, out_s\} \parallel \{P, E\}_i \xrightarrow{\tau} \{ps, in_s[i \mapsto \langle i, n_0, \delta_0 \rangle], out_s\} \parallel \{P, E[out_c \mapsto rs]\}_i} \end{array}$$

## CLIENTS

(READ)

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

(UPDATE)

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

(PUSH)

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

(PULL)

$$\frac{E.in_c \neq \emptyset \quad E.out_c \neq \emptyset \quad |E.rb| > 0}{\{pull(); P, E\}_i \xrightarrow{pull()} \{P, E[k \mapsto \mathbf{reducestate}(E.k, E.rb); rb \mapsto \epsilon; E.p \mapsto \mathbf{remove}(E.p, E.rb)]\}_i}$$

(CONFIRMED)

$$\{let\ x = \mathbf{confirmed}(); P, E\}_i \xrightarrow{\mathbf{confirmed}()} \{P[x \mapsto E.p = \epsilon \vee E.pb = \epsilon \vee E.tb = \epsilon \mapsto \mathbf{true}], E\}_i$$

(SEND)

$$\{P, E\}_i \xrightarrow{\mathbf{send}()} \{P, E[p \mapsto E.p \cdot E.pb; pb \mapsto \epsilon; out_c \mapsto out_c \cdot \langle i, E.n, E.pb \rangle]\}_i$$

(RECEIVE)

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

(DROP\_CONNECTION)

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

notify ::  $\mathcal{N}^* \times (\mathcal{N} \rightarrow GSSegment^* \cup GSPrefix) \times GSSegment$

notify( $\{b\}$ ,  $out_s$ ,  $gs$ ) =  $out_s(b) = gss \cdot gs$

notify( $b \cdot bs$ ,  $out_s$ ,  $gs$ ) =  $out_s(b) = gss \cdot gs \wedge notify(bs, out_s, gs)$

Esto no me convencí,  
discutir si hay una  
manera mejor

### 3 Equivalence

We have introduced an abstract GSP protocol and a robust streaming server-client implementation of GSP. There exists a relation represented by  $\triangleleft [\text{ref}]$  which relates state and delta objects to the update sequences.

- On  $\Delta \times \text{Update}^*$ , let  $\triangleleft$  be the smallest relation such that (1)  $\text{emptydelta} \triangleleft []$ , and (2)  $d \triangleleft a$  implies  $\mathbf{append}(d, u) \triangleleft a \cdot u$  for all updates  $u$ , and (3)  $d_1 \triangleleft a_1 \dots d_n \triangleleft a_n$  implies  $\mathbf{reduce}(d_1 \dots d_n) \triangleleft a_1 \dots a_n$  and (4)  $d_1 \triangleleft a_1 \dots d_n \triangleleft a_n$  implies  $\mathbf{remove}(d_1 \dots d_n, d_n) \triangleleft a_1 \dots a_{n-1}$ .

El (4) lo agregamos  
nosotros. Se aclara?  
Cómo?

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

Next theorem states one of most important result of the paper, saying that abstract GSP protocol and the implementation of GSP are weak bisimulation equivalent (or weakly bisimilar).

**Theorem 3.1.** *Let  $A$  be a system from abstract GSP Protocol, defined as then both system are weakly bisimilar; written  $A \approx C$ , if  $(A, C) \in \mathcal{R}$ , where  $\mathcal{R}$  be a binary relation defined as  $\{(\|i \in I_{\{0..n\}} A_i \| S_A), (\|i \in I_{\{0..n\}} C_i \| S_C) \mid \forall i \in \mathbb{N}, \forall r : Read, A_i = \{P_i, k_i, \rho_i, \alpha_i, \sigma_i, j_i\} \wedge C_i = \{P_i, E_i\} \text{ and the conjunction of the following properties:}$*

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

Ac' debería decir que  $\mathcal{R}$  cumple las propiedades de weak bisimulation. Cómo lo escribo mejor?

*Proof.* The proof follows by induction on the length of the derivation  $\rightarrow^*$ . We use  $\tilde{A}$  and  $\tilde{C}$  to denote clients, except client  $i^{th}$ , interacting concurrently, i.e.,  $\|j \in I_{\{0..n\}} - i A_j$  or  $C_j$  respectively.

- **n=0.** Then  $A = \{0, \emptyset, \emptyset, \emptyset, \emptyset, 0\}_i \| \tilde{A} \| S_A$  and  $C = \{E, initialstate\}_i$ . In particular,
  1. The operation  $rvalue$  in the abstract language is performing on empty sequences, hence, the result is undefined. Moreover the parametric function  $read$  tries to get a result on a  $initialstate$ , then the result returned is undefined.
  2. It is easy to see, due to property (1) of  $\triangleleft$  between States and Updates.
  3. It is guaranteed by property (1) of  $\triangleleft$  between Deltas and Updates.
  4. Analogous to property 2.
  5. Analogous to property 3.
  6. Precondition of implication is *false*.
  7. Precondition of both implications are *false*.
  8.  $\emptyset \in \emptyset$ .
- **n=k+1.**  $\forall (A, C) \in \mathcal{R}$ 
  - **rule (T-READ).** If  $A \xrightarrow{read(r)}_i A'$ , then by rule (T-READ),  $A$  must be the following term:  $\{let x = read(r); P, k, \rho, \alpha, \sigma, j\}_i \| \tilde{A} \| S_A$ , therefore, the client  $i^{th}$  is only one who changes. So that,  $A'$  will be  $\{P[x \mapsto v], k, \rho, \alpha, \sigma, j\}_i \| \tilde{A} \| S_A$ . As  $(A, C)$  belongs  $\mathcal{R}$  then  $C$  must be  $\{let x = read(r); P, E\}_i \| \tilde{C} \| S_C$ , and the read transition is the only one that  $C$  could perform. Then, looking at the rule,  $C' = \{P[x \mapsto v], E\}_i \| \tilde{C} \| S_C$ . We prove that  $(A', C') \in \mathcal{R}$  if nine properties introduced are keeping.

1. It is easy to see, since  $P$  is the same in both clients and the value got back by each read operations is equals because of  $(A, C) \in \mathcal{R}$  then property 1 is worth.
  2. to 9. Do not change.
- **rule (T-UPDATE).** If  $A \xrightarrow{\text{update}(u)}_i A'$ , then by rule (T-UPDATE),  $A$  must be the following term:  $\{\text{update}(u); P, k, \rho, \alpha, \sigma, j\}_i \parallel \tilde{A} \parallel S_A$ , therefore, the client  $i^{th}$  is only one who changes. So that,  $A'$  will be  $\{P, k, \rho, \alpha \cdot u, \sigma, j\}_i \parallel \tilde{A} \parallel S_A$ . We know that  $(A, C)$  belongs  $\mathcal{R}$ , then  $C$  must be  $\{\text{update}(u); P, E\}_i \parallel \tilde{C} \parallel S_C$ , and the update transition is the only one that  $C$  could perform. Then,  $C' = \{P, E[tb \mapsto \text{append}(E.tb, u)]\}_i \parallel \tilde{C} \parallel S_C$ . We prove that  $(A', C') \in \mathcal{R}$  if nine properties introduced are keeping.
    1. States  $rvalue(r, \text{flatten}(S_A[0..k_i - 1] \cdot \rho_i) \cdot \alpha_i) = \text{read}(r, \text{apply}(E_i.k, E_i.p \cdot E_i.pb \cdot E_i.tb))$  however after applying rules, the transaction buffers change. Then, performing a read action in abstract GSP,  $rvalue(r, \text{flatten}(S_A[0..k_i - 1] \cdot \rho_i) \cdot \alpha_i \cdot u)$  and the implementation,  $\text{read}(r, \text{apply}(E_i.k, E_i.p \cdot E_i.pb \cdot \text{append}(E.tb, u)))$ . We rename  $\alpha \cdot u$  by  $\alpha'$  and  $\text{append}(E.tb, u)$  by  $E_i.tb'$ . If we show that  $\forall u : \text{Update}$  such that  $\text{append}(E.tb, u) \triangleleft \alpha \cdot u$ , we will prove that  $(A', C') \in \mathcal{R}$ . By property 3, we know that  $E_i.tb \triangleleft [\alpha_i]$ , using the property of relation between Delta and Update,  $\text{append}(E.tb, u) \triangleleft \alpha \cdot u$ .
    2. to 9. Do not change.
  - **rule (T-PUSH).** If  $A \xrightarrow{\text{push}()}_i A'$ , then by rule (T-PUSH),  $A$  must be the following term:  $\{\text{push}(); P, k, \rho, \alpha, \sigma, j\}_i \parallel \tilde{A} \parallel S_A$ , therefore, the client  $i^{th}$  is only one who changes. So that,  $A'$  will be  $\{P, k, \rho \cdot [\alpha], \alpha \cdot u, \sigma \cdot [\alpha], j\}_i \parallel \tilde{A} \parallel S_A$ . We know that  $(A, C)$  belongs  $\mathcal{R}$ , then  $C$  must be  $\{\text{push}(); P, E\}_i \parallel \tilde{C} \parallel S_C$ , and the push transition is the only one that  $C$  could perform. Then, after applying the rule,  $C' = \{P, E[pb \mapsto \text{reduce}(E.pb \cdot E.tb); tb \mapsto \varepsilon; n \mapsto E.n + 1]\}_i \parallel \tilde{C} \parallel S_C$ . We prove that  $(A', C') \in \mathcal{R}$  if nine properties introduced are keeping.
    1. We have to prove that after push transitions, the operations  $rvalue$  and  $read$  get back the same value. Then  $rvalue(r, \text{flatten}(S_A[0..k_i - 1] \cdot \rho_i) \cdot \alpha_i) = rvalue(r, \text{flatten}(S_A[0..k_i - 1] \cdot (\rho_i \cdot [\alpha_i])) \cdot \varepsilon)$  by Lemma XX besides  $\text{read}(r, \text{curstate}(E_i.k, E_i.p, E_i.pb, E_i.tb)) = \text{read}(r, \text{curstate}(E_i.k, E_i.p, \text{reduce}(E.pb \cdot E.tb), \varepsilon))$  by Lemma YY.
    2. It does not change.
    3.  $E_i.tb = \varepsilon$  and  $\alpha_i = \varepsilon$ . Using the property of the relation  $\triangleleft$ ,  $\varepsilon \triangleleft \varepsilon$
    4. It does not change.
    5. It does not change.
    6. States  $E_i.p \cdot E_i.pb \triangleleft \rho_i$  besides by 3.  $E_i.tb \triangleleft [\alpha_i]$ . Finally, applying the property of the relation  $\triangleleft$ ,  $\text{reduce}(E_i.p \cdot E_i.pb \cdot E_i.tb) \triangleleft \rho_i \cdot [\alpha_i]$ .
    7. to 9. do not change.
  - **rule (T-PULL).** If  $A \xrightarrow{\text{pull}()}_i A'$ , then by rule (T-PULL),  $A$  must be the following term:  $\{\text{pull}(); P, k, \rho, \alpha, \sigma, j\}_i \parallel \tilde{A} \parallel S_A$ , therefore, the client  $i^{th}$  is only one who changes. So that,  $A'$  will be  $\{P, k + j, \rho \setminus S[k..k + j], \alpha, \sigma, 0\}_i \parallel \tilde{A} \parallel S_A$ . Due to  $(A, C)$  belongs  $\mathcal{R}$ , then  $C$  must be  $\{\text{pull}(); P, E\}_i \parallel \tilde{C} \parallel S_C$ , and the pull transition is the only one that  $C$  could perform. Then, after applying the rule,  $C' =$



$\{P, E[k \mapsto \text{reducestate}(E.k, E.rb); rb \mapsto \varepsilon; E.p \mapsto \text{remove}(E.p, E.rb)]\}_i \tilde{C} \parallel S_C$ .  
We prove that  $(A', C') \in \mathcal{R}$  if nine properties introduced are keeping.

1. We have to prove that after push transitions, the operations *rvalue* and *read* get back the same value. Then  $rvalue(r, \text{flatten}(S_A[0..k_i+j-1] \cdot (\rho \setminus S[k..k+j]))) \cdot \alpha_i = \text{read}(r, \text{curstate}(\text{reducestate}(E.k, E.rb), \text{remove}(E.p, E.rb), E.pb, E.tb)) = \text{read}(r, \text{apply}(\text{reducestate}(E.k, E.rb), \text{remove}(E.p, E.rb) \cdot E.pb \cdot E.tb))$ . On the one hand, property 5 states that  $E_i.known \triangleleft S_A[0..k_i-1]$ , on the other hand by property 4,  $E_i.rb[0].delta \triangleleft S_A[k_i] \dots E_i.rb[j-1].delta \triangleleft S_A[k_i+j-1]$ , after, using the property of the relation  $\triangleleft$  between state and updates, we know that  $\text{apply}(E_i.k, E_i.rb[0].delta \dots E_i.rb[j-1].delta) \triangleleft S_A[0..k_i-1] \cdot \dots \cdot S_A[k_i+j-1]$ . As you note, the left side is *reducestate*'s definition, then  $\text{reducestate}(E.k, E.rb)u \triangleleft S_A[0..k_i+j-1]$ . Next, property by 6., we know that  $E_i.p \cdot E_i.pb \triangleleft \rho_i$ , if we apply the extra axiom of  $\triangleleft$ , we will get that  $\text{remove}(E_i.p \cdot E_i.pb, E_i.rb[0].delta) \triangleleft \rho \setminus S[k]$ , so that,  $\text{remove}(E_i.p \cdot E_i.pb, E_i.rb) \triangleleft \rho \setminus S[k..k+j]$ . Therefore it is guaranteed.
2. It does not change.
3. It does not change.
4.  $E_i.rb$  is  $\varepsilon$  and  $j$  is 0, therefore it property is guaranteed.
5. It does not change.
6. We know that  $E_i.p \cdot E_i.pb \triangleleft \rho_i$ , if we apply the extra axiom of  $\triangleleft$ , we will get that  $\text{remove}(E_i.p \cdot E_i.pb, E_i.rb[0].delta) \triangleleft \rho \setminus S[k]$ , so that,  $\text{remove}(E_i.p \cdot E_i.pb, E_i.rb) \triangleleft \rho \setminus S[k..k+j]$ .
7. to 9. do not change.

• **rule (T-CONFIRMED)** If  $A \xrightarrow{\text{confirmed}()}_i A'$ , then by rule (T-CONFIRMED),  $A$  must be the following term:  $\{\text{let } x = \text{confirmed}(); P, k, \rho, \alpha, \sigma, j\}_i \tilde{A} \parallel S_A$ , therefore, the client  $i^{th}$  is only one who changes. So that,  $A'$  will be  $\{P[x \mapsto \rho \neq [] \vee \alpha \neq \varepsilon], k, \rho, \alpha, \sigma, j\}_i \tilde{A} \parallel S_A$ . As  $(A, C)$  belongs  $\mathcal{R}$  then  $C$  must be  $\{\text{let } x = \text{confirmed}(); P, E\}_i \tilde{C} \parallel S_C$ , and the read transition is the only one that  $C$  could perform. Then, looking at the rule,  $C' = \{P[x \mapsto E.p = \varepsilon \vee E.pb = \varepsilon \vee E.tb = \varepsilon \mapsto \text{true}], E\}_i \tilde{C} \parallel S_C$ . We prove that  $(A', C') \in \mathcal{R}$  if nine properties introduced are keeping.

1. Both clients has the same program  $P$ . By 6. we know that the value of  $E.p \cdot E.pb$  is related to  $\rho$ , thereby, when  $E.p$  or  $E.pb$  have any elements then  $\rho$  also has it. Finally,  $E_i.tb$  has elements iff  $\alpha_i$  also has by property 3.
2. to 9. Do not change.

• **rule (T-PROCESS)** If  $A \xrightarrow{\tau}_i A'$ , then by rule (T-PROCESS),  $A$  must be the following term:  $\{P, k, \rho, \alpha, [\beta] \cdot \sigma, j\}_i \tilde{A} \parallel S_A$ , therefore, the client  $i^{th}$  is only one who changes. So that,  $A'$  will be  $\{P, k, \rho, \alpha, \sigma, j\}_i \tilde{C} \parallel S \cdot \beta$ . Property 9 states that for all  $\beta$  such that  $\beta \in \sigma_i$  then  $\beta \in E_i.out_c \cup S_C.in_s$ . Analyzing cases:

- \* If  $\beta \in E_i.out_c$  then, by rule (COMM-CLIENT-SERVER),  $C \xrightarrow{\tau}_i C'$ , so that,  $C' = \{P, E[out_c \mapsto rs]\}_i \tilde{C} \parallel \{ps, in_s[i \mapsto \langle i, n_0, \delta_0 \rangle], out_s\}_C$  and  $\delta_0 \triangleleft \beta$ . Performing a new internal action by rule (BATCH), i.e.,  $C' \xrightarrow{\tau}_i C''$ , then  $C'' = \{\text{apply}(ps, d), in_s, \text{notify}(\text{dom}(out_s), out_s, gs)\}_i \tilde{C} \parallel \{P, E\}_i$ . We have to prove that  $(A', C'')$  belongs  $\mathcal{R}$ .

1. Theirs terms do not change.

2. Our hypothesis is  $S_C.ps \triangleleft S_A$  besides  $\delta_0 \triangleleft \beta$ , then using the property of the relation  $\triangleleft$  between state and updates, we will get that  $apply(S_C.ps, \delta_0) \triangleleft S_A \cdot \beta$ .
3. It does not change.
4. It does not change.
5. It does not change.
6. It does not change.
7. It does not change.
8.  $S_A$  has incremented by one, so that, by Hypothesis,  $k+j+1 \leq |S_A| < |S_A| + 1$ , besides,  $\Leftrightarrow |E_i.in_c| + |S_C.out_s| > 0$  because of neither  $E_i.in_c$  nor  $S_C.out_s$  have changed.
9. By Hypothesis, for all element in  $\sigma$ , those belongs to  $E_i.out_c \cup S_C.in_s$ , in particular,  $\sigma$  without a element keeps these guarantee.
- \* If  $\beta \in E_i.in_s$  then, applying rule (BATCH), we can see that we are at above case.

## 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.

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

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  $so \triangleright_i v$  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  $\{(v, w) / \{v \mapsto S[m]_h\} \in OP \wedge \{w \mapsto S[n]_i\} \in OP \wedge m < n\}$ . A transition  $\xrightarrow{\alpha}_i$  denotes the fact that action  $\alpha$  is performed by client  $i$ .

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

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

$$\begin{array}{c}
\text{(E-UPDATE)} \\
\frac{N \xrightarrow{\text{update}(u^v)}_i N' \quad v \notin \text{dom}(\text{OP})}{\{N, \text{OP}, \text{SO}, \text{VIS}, \text{AR}\} \xrightarrow{\text{update}(u^v)}_i \{N', \text{OP} \cup \{v \mapsto \text{update}(u)\}, \text{SO} \triangleright_i v, \text{VIS}, \text{AR}\}}
\end{array}$$

#### 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,  $\rho_i$  and  $\alpha_i$  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  $\{x \mapsto \text{update}(u)_i\} \in \text{OP} \Rightarrow u^x \in S[0..k_i-1] \cdot \rho_i \cdot [\alpha_i]$*

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

- **n=0.** Then  $\text{OP}$  is  $\emptyset$ , so that antecedent is false, then the preposition is true.
- **n=k+1.** Then  $\{N, \emptyset, \emptyset, \emptyset, \emptyset\} \rightarrow^n \{N, \text{OP}, \text{SO}, \text{VIS}, \text{AR}\} \rightarrow_i \{x \mapsto \text{update}(u)_i\} \in \text{OP} \Rightarrow u^x \in S[0..k_i-1] \cdot \rho_i \cdot [\alpha_i]$ . 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  $\{x \mapsto \text{update}(u)_i\} \in \text{OP}$ , then by inductive hypothesis  $u^x \in S[0..k_i-1] \cdot \rho_i \cdot [\alpha_i]$ .  
When  $N \xrightarrow{\text{read}(r)}_i N'$ ,  $k_i$ ,  $\rho_i$ ,  $\alpha_i$  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  $u^x \in S[0..k_i-1] \cdot \rho_i \cdot [\alpha_i] \cdot u_i^v$  too.
    - \*  $x = v$ . When  $N \xrightarrow{\text{update}(u^v)}_i N'$ ,  $u^v \in \alpha'_i$  (with  $\alpha'_i$  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  $\text{OP}$  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  $N \xrightarrow{\text{push}()}_i N'$ ,  $k'_i = k_i$ ,  $\rho'_i = \rho_i \cdot [\alpha_i]$  and  $\alpha'_i = \varepsilon$ .
  - **rule (E-PULL).** As  $\text{OP}$  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  $\text{OP}$ ,  $k_i$ ,  $\rho_i$  and  $\alpha_i$  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  $\{N_0, \emptyset, \emptyset, \emptyset, \emptyset\} \rightarrow^* \{N, OP, SO, VIS, AR\}$  then  $SO_R \cap (\mathbb{U} \times \mathbb{R}) \subseteq VIS$

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

- **n=0.** In particular  $OP$  and  $VIS$  are  $\emptyset$ , so that  $\emptyset \subseteq \emptyset$ .
- **n=k+1.** Then  $\{N_0, \emptyset, \emptyset, \emptyset, \emptyset\} \rightarrow^n \{N, OP, SO, VIS, AR\} \xrightarrow{\alpha}_i \{N', OP', SO', VIS', AR\}$ . 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  $SO_R \cup \{(w, v) / \{w \mapsto update(u)_j\}\}$ .
  - $VIS' = VIS \cup \{(x, v) / \{x \mapsto update(u)_h\} \in OP \wedge u^x \in S[0..k-1] \cdot p \cdot [\alpha]\}$ .

By inductive hypothesis,  $SO_R \cup (\mathbb{U} \times \mathbb{R}) \subseteq VIS$ . We only have to prove that  $\{(w, v) / \{w \mapsto update(u)_j\} \in OP\} \subseteq \{(x, v) / \{x \mapsto update(u)_h\} \in OP \wedge u^x \in S[0..k-1] \cdot p \cdot [\alpha]\}$ . When  $j = h$  and  $w = x$ , we can use Lemma 4.1. So that, we have proved that  $SO_R' \cap (\mathbb{U} \times \mathbb{R}) \subseteq VIS'$ .

**Theorem 4.2 (MONOTONIC READ).** Let  $SO_R$  the second component of the relation  $SO$  and  $VIS$  a visibility relation, if  $\{N_0, \emptyset, \emptyset, \emptyset, \emptyset\} \rightarrow^* \{N, OP, SO, VIS, AR\}$  then  $(VIS; SO_R) \cap (\mathbb{U} \times \mathbb{R}) \subseteq VIS$

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

- **n=0.** In particular  $SO_R$  and  $VIS$  are  $\emptyset$ , so that  $\emptyset \subseteq \emptyset$ .
- **n=k+1.** Then  $\{N_0, \emptyset, \emptyset, \emptyset, \emptyset\} \rightarrow^n \{N, OP, SO, VIS, AR\} \xrightarrow{\alpha}_i \{N', OP', SO', VIS', AR\}$ . Let  $R$  be a composition of relations. We shall say that if  $(x, y) \in R$  iff  $\exists y \in \mathbb{V}$  such that  $(x, y) \in VIS' \wedge (y, z) \in SO_R'$ . 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  $(y, z) \in SO_R$ , 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 4.1, there exists an update  $u$  such that  $u^x \in S[0..k_i-1] \cdot p_i \cdot [\alpha_i]$ . 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 vertex associated an update action, then  $SO_R' = SO_R \cup \{(w, v) / \{w \mapsto read(r)\} \vee \{w \mapsto update(u)_j\}\}$ . As we are only interested in relations of  $update \times read$ , then  $(VIS'; SO_R') \cap (\mathbb{U} \times \mathbb{R}) \equiv (VIS; SO_R) \cap (\mathbb{U} \times \mathbb{R})$ . By inductive hypothesis we can prove that  $(VIS'; SO_R') \cap (\mathbb{U} \times \mathbb{R}) \subseteq VIS \subseteq VIS'$ . 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  $\{N_0, \emptyset, \emptyset, \emptyset, \emptyset\} \rightarrow^* \{N, OP, SO, VIS, AR\}$  then **acyclic** $(SO_R \cup VIS)^+$ .

*Proof.* Since  $VIS$  is acyclic and  $SO_R \cap (\mathbb{U} \times \mathbb{R}) \subseteq VIS$  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,  $(SO_R \cup VIS)^+ \cap (\mathbb{U} \times \mathbb{R}) \subseteq VIS$ .

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

- **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:  $\forall (a, b) \mid (a, b \in \mathbb{V} \Rightarrow (a, b) \in (SO_R \cup VIS)^{k+1} \cap (\mathbb{U} \times \mathbb{R})) \Rightarrow (a, b) \in VIS$ .

Assume  $(a, x_1), (x_1, x_2), \dots, (x_{k-1}, x_k), (x_k, b)$  are relations from  $(SO_R \cup VIS)^{k+1}$ . Then  $(a, x_1), (x_1, x_2), \dots, (x_{k-1}, x_k)$  are relations from  $(SO_R \cup VIS)^k$ . By the induction hypothesis,  $(a, x_k) \in (SO_R \cup VIS)^k$ , and we also have  $(x_k, b) \in (SO_R \cup VIS)$ . Thus by the definition of  $(SO_R \cup VIS)^{k+1}$ ,  $(a, b) \in (SO_R \cup VIS)^{k+1}$ . Conversely, assume  $(a, b) \in (SO_R \cup VIS)^{k+1} = (SO_R \cup VIS)^k \circ (SO_R \cup VIS)$ . Then there is a vertex  $c \in \mathbb{V}$  such that  $(a, c) \in (SO_R \cup VIS)^k$  and  $(c, b) \in (SO_R \cup VIS)$ .

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  $(c, b) \in SO_R$ , they belong to the same client.
  - \* if  $(a, c) \in VIS$  and  $(c, b) \in SO_R$ , by Theorem 4.2,  $(a, b) \in VIS$ .
  - \* if  $(a, c) \in SO_R$  then by Theorem 4.1,  $(a, c) \in VIS$ . 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 a read action. In particular, if  $(a, c) \in SO_R$ , they belong to the same client.
  - \* if  $(a, c) \in SO_R$  and  $(c, b) \in VIS$ , by Theorem Monotonic Writes (Falta probar!),  $(a, b) \in VIS$ .
  - \* if  $(a, c) \in VIS \dots VER$ .

**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  $(\mathbb{U} \times \mathbb{U}) \cap (SO_R \cup VIS)^+ - SO_R \subseteq AR$ .

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

- **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  $(\text{Update} \times \text{Update})$ . We have two possible cases:

- $c$  is a read action. It means that  $(c,b)$  only can be in  $\text{SO}_R$  because  $\text{VIS}$  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:

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

Environment has a system  $N$  with two clients and a message queue  $S$  without updates. Relation sets  $\text{OP}, \text{SO}, \text{VIS}, \text{AR}$  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  $\text{update}(aList.add('a'))$  or  $\text{update}(aList.add('b'))$ . If the first communication takes place over  $\text{update}(aList.add('a'))$ , then the system evolves as follows:

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

The action  $\text{update}(aList.add('a'))$  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  $\text{so}$ . Now, Client 1 performs  $\text{push}()$ :

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

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.

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

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

$$\begin{aligned}
E & \xrightarrow{\text{update}(aList.add('b'))} \{ \{0, 0, [aList.add('a')], \epsilon, \epsilon, 0 \}_1 \parallel \\
& \{ \text{push}(); \text{let } x = \text{read}(aList); 0, \epsilon, [aList.add('b')], \epsilon, 0 \}_2 \parallel \\
& aList.add('a'), \emptyset, \{ (\{v_0, v_1\}, \{ (v_0, v_1) \}) \}, \emptyset, \emptyset \} \\
\\
& \xrightarrow{\text{push}()} \{ \{0, 0, [aList.add('a')], \epsilon, \epsilon, 0 \}_1 \parallel \\
& \{ \text{let } x = \text{read}(aList); 0, [aList.add('b')], \epsilon, [aList.add('b')], 0 \}_2 \parallel \\
& aList.add('a'), \emptyset, \{ (\{v_0, v_1\}, \{ (v_0, v_1) \}) \}, \emptyset, \emptyset \} \\
\\
& \xrightarrow{\tau} \{ \{0, 0, [aList.add('a')], \epsilon, \epsilon, 0 \}_1 \parallel \\
& \{ \text{let } x = \text{read}(aList); 0, [aList.add('b')], \epsilon, \epsilon, 0 \}_2 \parallel \\
& aList.add('a') \cdot [aList.add('b')], \emptyset, \{ (\{v_0, v_1\}, \{ (v_0, v_1) \}) \}, \emptyset, \{ (v_0, v_1) \} \} \\
\\
& \xrightarrow{\text{read}(aList)} \{ \{0, 0, [aList.add('a')], \epsilon, \epsilon, 0 \}_1 \parallel \\
& \{ 0[x \mapsto [b']], 0, [aList.add('b')], \epsilon, \epsilon, 0 \}_2 \parallel aList.add('a') \cdot [aList.add('b')], \\
& \emptyset, \emptyset, \emptyset \}
\end{aligned}$$

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  $(AR; VIS) \subseteq AR; VIS$ , 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.

$$\begin{array}{ll}
(\text{ENVIRONMENT}) & E ::= \{N, \text{OP}, \text{SO}, \text{VIS}, \text{AR}, \text{RB}, \text{TO}, \text{TC}\} \\
(\text{TRANSACTIONS WITH OVERLAPPING}) & \text{TO} ::= \emptyset \mid w^j \mapsto \mathcal{N}, \text{TO} \text{ with } w \in \mathbb{V} \ j \in \mathbb{N} \\
(\text{TRANSACTIONS CLOSED}) & \text{TC} ::= \epsilon \mid w^j \cdot \text{TC} \text{ with } w \in \mathbb{V} \ j \in \mathbb{N}
\end{array}$$

The term TO relates vertex from a client to set of vertex which have just not finished; TC denotes transaction closed.

### 5.1 Operational Semantic

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

The labeled transition system for the extension of GSP considers the following actions  $\beta$ :

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

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

#### ENVIRONMENT

(E-START-SYNC)

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

(E-END-SYNC)

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

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)

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

(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)

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

(T-END-SYNC)

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