# Flat Committed Join in Join

**Hernán Melgratti**

joint work with **Roberto Bruni** and **Ugo Montanari**

Dipartimento di Informatica - Università di Pisa

# Committed Join (cJoin)

- Join + primitives for negotiations
- Syntax:

| | |
|---|---|
| Processes | $P,Q ::= 0 \mid x\langle\tilde{y}\rangle \mid$ **def** $D$ **in** $P \mid P|Q$ |
| Definitions | $D,E ::= J \triangleright P \mid D \wedge E$ |
| Patterns | $J,K ::= x\langle\tilde{y}\rangle \mid J|K$ |



# Committed Join (cJoin)

- Join + primitives for negotiations
- Syntax:

| | |
|---|---|
| Messages | $M,N ::= 0 \mid x\langle\tilde{y}\rangle \mid M|N$ |
| Processes | $P,Q ::= 0 \mid x\langle\tilde{y}\rangle \mid$ **def** $D$ **in** $P \mid P|Q$ |
| Definitions | $D,E ::= J \triangleright P \mid D \wedge E$ |
| Patterns | $J,K ::= x\langle\tilde{y}\rangle \mid J|K$ |



# Committed Join (cJoin)

- Join + primitives for negotiations
- Syntax:

| | |
|---|---|
| Messages | $M,N ::= 0 \mid x\langle\tilde{y}\rangle \mid M|N$ |
| Processes | $P,Q ::= M \mid$ **def** $D$ **in** $P \mid P|Q$ |
| Definitions | $D,E ::= J \triangleright P \mid D \wedge E$ |
| Patterns | $J,K ::= x\langle\tilde{y}\rangle \mid J|K$ |



# Committed Join (cJoin)

- Join + primitives for negotiations
- Syntax:

| | |
|---|---|
| Messages | $M,N ::= 0 \mid x\langle\tilde{y}\rangle \mid M|N$ |
| Processes | $P,Q ::= M \mid$ **def** $D$ **in** $P \mid P|Q \mid$ ***abort*** $\mid$ **[P:Q]** |
| Definitions | $D,E ::= J \triangleright P \mid D \wedge E \mid J \blacktriangleright P$ |
| Patterns | $J,K ::= x\langle\tilde{y}\rangle \mid J|K$ |

Programmable abort

Negotiation

Compensation

Merge definition



# Committed Join (cJoin)

- Operational Semantics (CHAM Style):

$$0 \rightleftharpoons \quad \text{(heating and cooling)}$$
$$P|Q \rightleftharpoons P,Q$$
$$D \wedge E \rightleftharpoons D,E$$
$$\textbf{def } D \textbf{ in } P \rightleftharpoons D\sigma_{dn(D)}, P\sigma_{dn(D)} \quad range(\sigma) \text{ fresh}$$
$$J \triangleright P, J\sigma \rightarrow J \triangleright P, P\sigma \quad \text{(reaction)}$$

# Committed Join (cJoin)

- Operational Semantics (CHAM Style):

$$
\begin{array}{rcl}
0 & \rightleftarrows & \\
P|Q & \rightleftarrows & P,Q \\
D \wedge E & \rightleftarrows & D,E \\
\textbf{def}\ D\ \textbf{in}\ P & \rightleftarrows & D\sigma_{dn(D)}\ ,\ P\sigma_{dn(D)} \quad \text{range}(\sigma)\ \text{fresh} \\
J \triangleright P,\ J\sigma & \rightarrow & J \triangleright P,\ P\sigma \\
[P:Q] & \rightleftarrows & \{[\ P\ ,\ \llcorner Q \lrcorner\ ]\}
\end{array}
$$

Contract P evolves in isolation

Compensation Q is kept frozen



# Committed Join (cJoin)

- Operational Semantics (CHAM Style):

$$
\begin{array}{rcl}
0 & \rightleftarrows & \\
P|Q & \rightleftarrows & P,Q \\
D \wedge E & \rightleftarrows & D,E \\
\textbf{def}\ D\ \textbf{in}\ P & \rightleftarrows & D\sigma_{dn(D)}\ ,\ P\sigma_{dn(D)} \quad \text{range}(\sigma)\ \text{fresh} \\
J \triangleright P,\ J\sigma & \rightarrow & J \triangleright P,\ P\sigma \\
[P:Q] & \rightleftarrows & \{[\ P\ ,\ \llcorner Q \lrcorner\ ]\} \\
\{[\ M|\textbf{def}\ D\ \textbf{in}\ 0\ ,\ \llcorner Q \lrcorner\ ]\} & \rightarrow & M
\end{array}
$$

Global Resources

Commit



# Committed Join (cJoin)

- Operational Semantics (CHAM Style):

$$
\begin{array}{rcl}
0 & \rightleftarrows & \\
P|Q & \rightleftarrows & P,Q \\
D \wedge E & \rightleftarrows & D,E \\
\textbf{def}\ D\ \textbf{in}\ P & \rightleftarrows & D\sigma_{dn(D)}\ ,\ P\sigma_{dn(D)} \quad \text{range}(\sigma)\ \text{fresh} \\
J \triangleright P,\ J\sigma & \rightarrow & J \triangleright P,\ P\sigma \\
[P:Q] & \rightleftarrows & \{[\ P\ ,\ \llcorner Q \lrcorner\ ]\} \\
\{[\ M|\textbf{def}\ D\ \textbf{in}\ 0\ ,\ \llcorner Q \lrcorner\ ]\} & \rightarrow & M \\
\{[\ \textit{abort}\ |P\ ,\ \llcorner Q \lrcorner\ ]\} & \rightarrow & Q
\end{array}
$$

Compensation on Abort



# Committed Join (cJoin)

- Operational Semantics (CHAM Style):

$$
\begin{array}{rcl}
0 & \rightleftarrows & \\
P|Q & \rightleftarrows & P,Q \\
D \wedge E & \rightleftarrows & D,E \\
\textbf{def}\ D\ \textbf{in}\ P & \rightleftarrows & D\sigma_{dn(D)}\ ,\ P\sigma_{dn(D)} \quad \text{range}(\sigma)\ \text{fresh} \\
J \triangleright P,\ J\sigma & \rightarrow & J \triangleright P,\ P\sigma \\
[P:Q] & \rightleftarrows & \{[\ P\ ,\ \llcorner Q \lrcorner\ ]\} \\
\{[\ M|\textbf{def}\ D\ \textbf{in}\ 0\ ,\ \llcorner Q \lrcorner\ ]\} & \rightarrow & M \\
\{[\ \textit{abort}\ |P\ ,\ \llcorner Q \lrcorner\ ]\} & \rightarrow & Q \\
J_1|\ldots|J_n \blacktriangleright P,\ \otimes_i\{[\ J_i\sigma,\ S_i,\ \llcorner Q_i \lrcorner\ ]\} & \rightarrow & J_1|\ldots|J_n \blacktriangleright P,\ \{[\otimes_i S_i,\ P\sigma,\ \llcorner \Pi_i Q_i \lrcorner\ ]\}
\end{array}
$$

Merge n ongoing contracts



# Example: Mailing List

$\mathbf{ML} \equiv \text{MailingList}\langle k \rangle \triangleright \mathbf{MLDef}$



# Example: Mailing List

$\mathbf{ML} \equiv \text{MailingList}\langle k \rangle \triangleright \mathbf{MLDef}$

$\mathbf{MLDef} \equiv \textbf{def}\ \ldots$

$\textbf{in}\ k\langle \text{add, tell, close} \rangle\ |\ \text{lst}\langle \text{nil} \rangle$

## Example: Mailing List

**ML** ≡ MailingList⟨k⟩ ▷ **MLDef**

**MLDef** ≡ **def** …
∧ lst⟨y⟩ | add⟨x⟩ ▷ …
∧ lst⟨y⟩ | tell⟨v⟩ ▷ …
∧ lst⟨y⟩ | close⟨ ⟩ ▷ …
**in** k⟨add, tell, close⟩ | lst⟨nil⟩



## Example: Mailing List

**ML** ≡ MailingList⟨k⟩ ▷ **MLDef**

**MLDef** ≡ **def** …
∧ lst⟨y⟩ | add⟨x⟩ ▷ **def** z⟨v,w⟩ ▸ x⟨v⟩ | y⟨v,w⟩ **in** lst⟨z⟩
∧ lst⟨y⟩ | tell⟨v⟩ ▷ …
∧ lst⟨y⟩ | close⟨ ⟩ ▷ …
**in** k⟨add, tell, close⟩ | lst⟨nil⟩



## Example: Mailing List

**ML** ≡ MailingList⟨k⟩ ▷ **MLDef**

**MLDef** ≡ **def** …
∧ lst⟨y⟩ | add⟨x⟩ ▷ **def** z⟨v,w⟩ ▸ x⟨v⟩ | y⟨v,w⟩ **in** lst⟨z⟩
∧ lst⟨y⟩ | tell⟨v⟩ ▷ [**def** w⟨⟩ ▸ 0 **in** y⟨v,w⟩| lst⟨y⟩ **:** lst⟨y⟩]
∧ lst⟨y⟩ | close⟨ ⟩ ▷ …
**in** k⟨add, tell, close⟩ | lst⟨nil⟩



## Example: Mailing List

**ML** ≡ MailingList⟨k⟩ ▷ **MLDef**

**MLDef** ≡ **def** nil⟨v, w⟩ ▸ 0
∧ lst⟨y⟩ | add⟨x⟩ ▷ **def** z⟨v,w⟩ ▸ x⟨v⟩ | y⟨v,w⟩ **in** lst⟨z⟩
∧ lst⟨y⟩ | tell⟨v⟩ ▷ [**def** w⟨⟩ ▸ 0 **in** y⟨v,w⟩| lst⟨y⟩ **:** lst⟨y⟩]
∧ lst⟨y⟩ | close⟨ ⟩ ▷ …
**in** k⟨add, tell, close⟩ | lst⟨nil⟩



## Example: Mailing List

**ML** ≡ MailingList⟨k⟩ ▷ **MLDef**

**MLDef** ≡ **def** nil⟨v, w⟩ ▸ 0
∧ lst⟨y⟩ | add⟨x⟩ ▷ **def** z⟨v,w⟩ ▸ x⟨v⟩ | y⟨v,w⟩ **in** lst⟨z⟩
∧ lst⟨y⟩ | tell⟨v⟩ ▷ [**def** w⟨⟩ ▸ 0 **in** y⟨v,w⟩| lst⟨y⟩ **:** lst⟨y⟩]
∧ lst⟨y⟩ | close⟨ ⟩ ▷ 0
**in** k⟨add, tell, close⟩ | lst⟨nil⟩



## Implementing flat cJoin

- Goal
  - To implement cJoin in Join
- Key Points
  - Distributed commit of interacting negotiations as a global decision
  - Participants can join dynamically (participants statically unknown)
- First step
  - Consider flat negotiations
  - Use canonical form of processes
  - Encode canonical flat cJoin processes as Join processes

## Flat cJoin

- Negotiations cannot be nested
- Type system for cJoin Processes:
  - P: $\square_0$, P does not contain [_:_] at all
  - P: $\square_1$, P may contain [_:_] in the definitions
  - P: $\square_2$, P may have and generate flat negotiations
- Join Processes have type $\square_0$
- Subject Reduction holds for $\square_0$ and $\square_2$
- Flat cJoin: The sub-calculus of all P:$\square_2$



## Canonical Flat cJoin

- Inspired by the basic shapes of ZS nets
- Few elementary definition patterns

| | | | |
|---|---|---|---|
| **Open** | $x\langle\hat{y}\rangle \triangleright P$ | & P: $\square_2$ | & count(P) = 1 |
| **Ord-Mov** | $x\langle\hat{y}\rangle \triangleright P$ | & P: $\square_1$ | & count(P) ≤ 2 |
| **Merge-Mov** | $x\langle\hat{y}\rangle \blacktriangleright P$ | & P: $\square_0$ | & count(P) ≤ 2 |
| **Ord-Join** | $x\langle\hat{y}_1\rangle \mid x\langle\hat{y}_2\rangle \triangleright P$ | & P: $\square_1$ | & count(P) = 1 |
| **Merge-Join** | $x\langle\hat{y}_1\rangle \mid \ldots \mid x\langle\hat{y}_n\rangle \blacktriangleright P$ | & P: $\square_0$ | & count(P) = 1 |

- Any flat process can be written in canonical form



## Canonical Form: Example

**def** nil⟨v, w⟩ ▸ 0
∧ lst⟨y⟩ | add⟨x⟩ ▹ **def** z⟨v,w⟩ ▸ x⟨v⟩ | y⟨v,w⟩ **in** lst⟨z⟩
∧ lst⟨y⟩ | tell⟨v⟩ ▹ [**def** w⟨⟩ ▸ 0 **in** y⟨v,w⟩| lst⟨y⟩ **:** lst⟨y⟩]
∧ lst⟨y⟩ | close⟨ ⟩ ▹ 0
**in** k⟨add, tell, close⟩ | lst⟨nil⟩

**Has type** $\square_2$

∧ lst⟨y⟩ | tell⟨x⟩ ▹ a⟨y,x⟩
∧ a⟨y,x⟩ ▹ [**def** w⟨⟩ ▸ 0 **in** y⟨v,w⟩| lst⟨y⟩ **:** lst⟨y⟩]

**Has count = 2**

∧ a⟨y,x⟩ ▹ [**def** w⟨⟩ ▸ 0 **in def** b⟨v,w,l,y⟩ ▹ y⟨v,w⟩| l⟨y⟩ **in** b⟨v,w,lst,y⟩ **:** lst⟨y⟩]



## Encoding: Main Idea

- Any message in a negotiation runs in a thread, which is managed by a coordinator
- Coordinators perform a D2PC protocol [BLM2002].
  - A variant of the decentralized 2PC with a finite but unknown number of participants
  - When a participant P is ready to commit it has only a partial knowledge of the whole set of participants
    - Only those who directly cooperated with P
  - To commit P must contact all its neighbors and learn the identity of other participants from them



## Encoding: D2PC

- Every participant P acts as coordinator
  - During the transaction P builds its own synchronization set $L_P$ of cooperating agents
  - When P is ready to commit, P asks readiness to processes in $L_P$ (if empty P was isolated and can commit)
    - In doing so, P sends them the set $L_P$
  - Other participants will send to P
    - either a successful reply with their own synchronization sets
    - or a failure message
      - (in this case, failure is then propagated)
  - Successful replies are added to $L_P$
  - The protocol terminates when $L_P$ is transitively closed



## Encoding a negotiation

[**def** z⟨⟩ ▹ 0 **in** z⟨⟩ **:** 0]

## Encoding a negotiation

A negotiation
with one thread

[**def** z⟨⟩ ▹ 0 **in** z⟨⟩ **:** 0]



## Encoding a negotiation

- put: to notify thread completion
- abt: to notify thread abortion
- lock: to receive partners' confirmations
- state: to set the initial compensation

A negotiation
with one thread

[**def** z⟨⟩ ▹ 0 **in** z⟨⟩ **:** 0]

A coordinator D
to manage z⟨⟩

**def D**



## Encoding a negotiation

- put: to notify thread completion
- abt: to notify thread abortion
- lock: to receive partners' confirmations
- state: to set the initial compensation

[**def** z⟨⟩ ▹ 0 **in** z⟨⟩ **:** 0]

Compensation

**def D**



## Encoding a negotiation

- put: to notify thread completion
- abt: to notify thread abortion
- lock: to receive partners' confirmations
- state: to set the initial compensation

[**def** z⟨⟩ ▹ 0 **in** z⟨⟩ **:** 0]

Compensation

Set initial
compensation

**def D** ∧ cmp⟨⟩ ▹ 0
**in** state ⟨{cmp}⟩



## Encoding a negotiation

- put: to notify thread completion
- abt: to notify thread abortion
- lock: to receive partners' confirmations
- state: to set the initial compensation

[**def** z⟨⟩ ▹ 0 **in** z⟨⟩ **:** 0]

z runs in a
managed thread

**def D** ∧ cmp⟨⟩ ▹ 0
**in** state ⟨{cmp}⟩



## Encoding a negotiation

- put: to notify thread completion
- abt: to notify thread abortion
- lock: to receive partners' confirmations
- state: to set the initial compensation

[**def** z⟨⟩ ▹ 0 **in** z⟨⟩ **:** 0]

z runs in a
managed thread

z carries the ports
of its coordinator

**def D** ∧ cmp⟨⟩ ▹ 0
**in** state ⟨{cmp}⟩ |
**def** …
**in** z⟨put, abt, {lock}⟩

## Encoding a negotiation

- put: to notify thread completion
- abt: to notify thread abortion
- lock: to receive partners' confirmations
- state: to set the initial compensation

[**def** z⟨⟩ ▹ 0 **in** z⟨⟩ **:** 0]

**It consumes managed msgs**

**def D** ∧ cmp⟨⟩ ▹ 0
**in** state ⟨{cmp}⟩ |
**def** …
**in** z⟨put, abt, {lock}⟩



## Encoding a negotiation

- put: to notify thread completion
- abt: to notify thread abortion
- lock: to receive partners' confirmations
- state: to set the initial compensation

[**def** z⟨⟩ ▹ 0 **in** z⟨⟩ **:** 0]

**It consumes managed msgs**

z carries the ports of its coordinator

**def D** ∧ cmp⟨⟩ ▹ 0
**in** state ⟨{cmp}⟩ |
**def** z⟨p, a, l⟩ ▹ …
**in** z⟨put, abt, {lock}⟩



## Encoding a negotiation

- put: to notify thread completion
- abt: to notify thread abortion
- lock: to receive partners' confirmations
- state: to set the initial compensation

[**def** z⟨⟩ ▹ 0 **in** z⟨⟩ **:** 0]

**The thread ends**

**def D** ∧ cmp⟨⟩ ▹ 0
**in** state ⟨{cmp}⟩ |
**def** z⟨p, a, l⟩ ▹ …
**in** z⟨put, abt, {lock}⟩



## Encoding a negotiation

- put: to notify thread completion
- abt: to notify thread abortion
- lock: to receive partners' confirmations
- state: to set the initial compensation

[**def** z⟨⟩ ▹ 0 **in** z⟨⟩ **:** 0]

**The thread ends**

D can commit

**def D** ∧ cmp⟨⟩ ▹ 0
**in** state ⟨{cmp}⟩ |
**def** z⟨p, a, l⟩ ▹ p⟨l, ∅, ∅⟩
**in** z⟨put, abt, {lock}⟩



## Encoding an abort

- put: to notify thread completion
- abt: to notify thread abortion
- lock: to receive partners' confirmations
- state: to set the initial compensation

[**def** z⟨⟩ ▹ abort **in** z⟨⟩ **:** 0]

**The thread aborts**

**def D** ∧ cmp⟨⟩ ▹ 0
**in** state ⟨{cmp}⟩ |
**def** z⟨p, a, l⟩ ▹ …
**in** z⟨put, abt, {lock}⟩



## Encoding an abort

- put: to notify thread completion
- abt: to notify thread abortion
- lock: to receive partners' confirmations
- state: to set the initial compensation

[**def** z⟨⟩ ▹ abort **in** z⟨⟩ **:** 0]

**The thread aborts**

D aborts

**def D** ∧ cmp⟨⟩ ▹ 0
**in** state ⟨{cmp}⟩ |
**def** z⟨p, a, l⟩ ▹ a⟨⟩
**in** z⟨put, abt, {lock}⟩

## Encoding: Fork and Join

$x\langle\rangle \triangleright y\langle\rangle | z\langle\rangle$ ➡



## Encoding: Fork and Join

$x\langle\rangle \triangleright y\langle\rangle | z\langle\rangle$ ➡ $x\langle p,a,l\rangle \triangleright \mathbf{def}\ D_1 \wedge D_2$



## Encoding: Fork and Join

$x\langle\rangle \triangleright y\langle\rangle | z\langle\rangle$ ➡ $x\langle p,a,l\rangle \triangleright \mathbf{def}\ D_1 \wedge D_2$

$\mathbf{in}\ y\langle put_1, abt_1, l \cup \{lock_1, lock_2\}\rangle$

$|\ z\langle put_2, abt_2, l \cup \{lock_1, lock_2\}\rangle$



## Encoding: Fork and Join

$x\langle\rangle \triangleright y\langle\rangle | z\langle\rangle$ ➡ $x\langle p,a,l\rangle \triangleright \mathbf{def}\ D_1 \wedge D_2$

$\mathbf{in}\ y\langle put_1, abt_1, l \cup \{lock_1, lock_2\}\rangle$

$|\ z\langle put_2, abt_2, l \cup \{lock_1, lock_2\}\rangle$

$|\ p\langle l \cup \{lock_1, lock_2\}, \{abt_1, abt_2\}, \emptyset\rangle$



## Encoding: Fork and Join

$x\langle\rangle \triangleright y\langle\rangle | z\langle\rangle$ ➡ $x\langle p,a,l\rangle \triangleright \mathbf{def}\ D_1 \wedge D_2$

$\mathbf{in}\ y\langle put_1, abt_1, l \cup \{lock_1, lock_2\}\rangle$

$|\ z\langle put_2, abt_2, l \cup \{lock_1, lock_2\}\rangle$

$|\ p\langle l \cup \{lock_1, lock_2\}, \{abt_1, abt_2\}, \emptyset\rangle$

$|\ state_1\langle\{a, abt_2\}\rangle$

$|\ state_2\langle\{a, abt_1\}\rangle$



## Encoding: Fork and Join

$x\langle\rangle \triangleright y\langle\rangle | z\langle\rangle$ ➡ $x\langle p,a,l\rangle \triangleright \mathbf{def}\ D_1 \wedge D_2$

$\mathbf{in}\ y\langle put_1, abt_1, l \cup \{lock_1, lock_2\}\rangle$

$|\ z\langle put_2, abt_2, l \cup \{lock_1, lock_2\}\rangle$

$|\ p\langle l \cup \{lock_1, lock_2\}, \{abt_1, abt_2\}, \emptyset\rangle$

$|\ state_1\langle\{a, abt_2\}\rangle$

$|\ state_2\langle\{a, abt_1\}\rangle$

$x\langle\rangle | y\langle\rangle \triangleright z\langle\rangle$ ➡

## Encoding: Fork and Join

$x\langle\rangle \triangleright y\langle\rangle | z\langle\rangle$ ➡ $x\langle p,a,l\rangle \triangleright$ **def** $D_1 \wedge D_2$
**in** $y\langle put_1, abt_1, l \cup \{lock_1, lock_2\}\rangle$
$| \; z\langle put_2, abt_2, l \cup \{lock_1, lock_2\}\rangle$
$| \; p\langle l \cup \{lock_1, lock_2\}, \{abt_1, abt_2\}, \emptyset\rangle$
$| \; state_1\langle\{a, abt_2\}\rangle$
$| \; state_2\langle\{a, abt_1\}\rangle$

$x\langle\rangle | y\langle\rangle \triangleright z\langle\rangle$ ➡ $x\langle p_1,a_1,l_1\rangle | y\langle p_2,a_2,l_2\rangle \triangleright$ **def** $D$



## Encoding: Fork and Join

$x\langle\rangle \triangleright y\langle\rangle | z\langle\rangle$ ➡ $x\langle p,a,l\rangle \triangleright$ **def** $D_1 \wedge D_2$
**in** $y\langle put_1, abt_1, l \cup \{lock_1, lock_2\}\rangle$
$| \; z\langle put_2, abt_2, l \cup \{lock_1, lock_2\}\rangle$
$| \; p\langle l \cup \{lock_1, lock_2\}, \{abt_1, abt_2\}, \emptyset\rangle$
$| \; state_1\langle\{a, abt_2\}\rangle$
$| \; state_2\langle\{a, abt_1\}\rangle$

$x\langle\rangle | y\langle\rangle \triangleright z\langle\rangle$ ➡ $x\langle p_1,a_1,l_1\rangle | y\langle p_2,a_2,l_2\rangle \triangleright$ **def** $D$
**in** $z\langle put, abt_1, l_1 \cup l_2 \cup \{lock\}\rangle$



## Encoding: Fork and Join

$x\langle\rangle \triangleright y\langle\rangle | z\langle\rangle$ ➡ $x\langle p,a,l\rangle \triangleright$ **def** $D_1 \wedge D_2$
**in** $y\langle put_1, abt_1, l \cup \{lock_1, lock_2\}\rangle$
$| \; z\langle put_2, abt_2, l \cup \{lock_1, lock_2\}\rangle$
$| \; p\langle l \cup \{lock_1, lock_2\}, \{abt_1, abt_2\}, \emptyset\rangle$
$| \; state_1\langle\{a, abt_2\}\rangle$
$| \; state_2\langle\{a, abt_1\}\rangle$

$x\langle\rangle | y\langle\rangle \triangleright z\langle\rangle$ ➡ $x\langle p_1,a_1,l_1\rangle | y\langle p_2,a_2,l_2\rangle \triangleright$ **def** $D$
**in** $z\langle put, abt_1, l_1 \cup l_2 \cup \{lock\}\rangle$
$| \; p_1\langle l_1 \cup l_2 \cup \{lock\}, \{abt, a_2\}, \emptyset\rangle$
$| \; p_2\langle l_1 \cup l_2 \cup \{lock\}, \{abt, a_1\}, \emptyset\rangle$



## Encoding: Fork and Join

$x\langle\rangle \triangleright y\langle\rangle | z\langle\rangle$ ➡ $x\langle p,a,l\rangle \triangleright$ **def** $D_1 \wedge D_2$
**in** $y\langle put_1, abt_1, l \cup \{lock_1, lock_2\}\rangle$
$| \; z\langle put_2, abt_2, l \cup \{lock_1, lock_2\}\rangle$
$| \; p\langle l \cup \{lock_1, lock_2\}, \{abt_1, abt_2\}, \emptyset\rangle$
$| \; state_1\langle\{a, abt_2\}\rangle$
$| \; state_2\langle\{a, abt_1\}\rangle$

$x\langle\rangle | y\langle\rangle \triangleright z\langle\rangle$ ➡ $x\langle p_1,a_1,l_1\rangle | y\langle p_2,a_2,l_2\rangle \triangleright$ **def** $D$
**in** $z\langle put, abt_1, l_1 \cup l_2 \cup \{lock\}\rangle$
$| \; p_1\langle l_1 \cup l_2 \cup \{lock\}, \{abt, a_2\}, \emptyset\rangle$
$| \; p_2\langle l_1 \cup l_2 \cup \{lock\}, \{abt, a_1\}, \emptyset\rangle$
$| \; state\langle\{a_1, a_2\}\rangle$



## Encoding: Names

[**def** $z\langle x\rangle \triangleright x\langle x\rangle$ **in** $z\langle y\rangle$: 0]



## Encoding: Names

[**def** $z\langle x\rangle \triangleright x\langle x\rangle$ **in** $z\langle y\rangle$: 0] $\rightarrow$ [**def** $z\langle x\rangle \triangleright x\langle x\rangle$ **in** $y\langle y\rangle$: 0]

## Encoding: Names

[**def** z⟨x⟩ ▷ x⟨x⟩ **in** z⟨y⟩: 0] → [**def** z⟨x⟩ ▷ x⟨x⟩ **in** y⟨y⟩: 0]
⇄* [y⟨y⟩ | **def** z⟨x⟩ ▷ x⟨x⟩ **in** 0: 0]

Workshop Finale Cometa – Dec, 2003

## Encoding: Names

[**def** z⟨x⟩ ▷ x⟨x⟩ **in** z⟨y⟩: 0] → [**def** z⟨x⟩ ▷ x⟨x⟩ **in** y⟨y⟩: 0]
⇄* [y⟨y⟩ | **def** z⟨x⟩ ▷ x⟨x⟩ **in** 0: 0]
→ y⟨y⟩

Workshop Finale Cometa – Dec, 2003

## Encoding: Names

[**def** z⟨x⟩ ▷ x⟨x⟩ **in** z⟨y⟩: 0] → [**def** z⟨x⟩ ▷ x⟨x⟩ **in** y⟨y⟩: 0]
⇄* [y⟨y⟩ | **def** z⟨x⟩ ▷ x⟨x⟩ **in** 0: 0]
→ y⟨y⟩

[**def** z⟨x⟩ ▷ x⟨x⟩ **in** z⟨z⟩: 0]

Workshop Finale Cometa – Dec, 2003

## Encoding: Names

[**def** z⟨x⟩ ▷ x⟨x⟩ **in** z⟨y⟩: 0] → [**def** z⟨x⟩ ▷ x⟨x⟩ **in** y⟨y⟩: 0]
⇄* [y⟨y⟩ | **def** z⟨x⟩ ▷ x⟨x⟩ **in** 0: 0]
→ y⟨y⟩

[**def** z⟨x⟩ ▷ x⟨x⟩ **in** z⟨z⟩: 0] → [**def** z⟨x⟩ ▷ x⟨x⟩ **in** z⟨z⟩: 0]

Workshop Finale Cometa – Dec, 2003

## Encoding: Names

[**def** z⟨x⟩ ▷ x⟨x⟩ **in** z⟨y⟩: 0] → [**def** z⟨x⟩ ▷ x⟨x⟩ **in** y⟨y⟩: 0]
⇄* [y⟨y⟩ | **def** z⟨x⟩ ▷ x⟨x⟩ **in** 0: 0]
→ y⟨y⟩

[**def** z⟨x⟩ ▷ x⟨x⟩ **in** z⟨z⟩: 0] → [**def** z⟨x⟩ ▷ x⟨x⟩ **in** z⟨z⟩: 0]
→ [**def** z⟨x⟩ ▷ x⟨x⟩ **in** z⟨z⟩: 0]
→ ...

Workshop Finale Cometa – Dec, 2003

## Encoding: Names

[**def** z⟨x⟩ ▷ x⟨x⟩ **in** z⟨y⟩: 0] → [**def** z⟨x⟩ ▷ x⟨x⟩ **in** y⟨y⟩: 0]
⇄* [y⟨y⟩ | **def** z⟨x⟩ ▷ x⟨x⟩ **in** 0: 0]
→ y⟨y⟩

[**def** z⟨x⟩ ▷ x⟨x⟩ **in** z⟨z⟩: 0] → [**def** z⟨x⟩ ▷ x⟨x⟩ **in** z⟨z⟩: 0]
→ [**def** z⟨x⟩ ▷ x⟨x⟩ **in** z⟨z⟩: 0]
→ ...

Definitions have different behaviours depending on received names

Workshop Finale Cometa – Dec, 2003

## Encoding: Names

- A definition is encoded as several definitions:

$$z\langle x\rangle \triangleright x\langle x\rangle \quad \Longrightarrow \quad \begin{cases} z^s\langle x, p, a, l\rangle \triangleright p\,\langle l, \emptyset, \{x\langle x\rangle\}\rangle \\ z^z\langle x, p, a, l\rangle \triangleright x\,\langle x, p, a, l\rangle \end{cases}$$



## Encoding: Names

- A definition is encoded as several definitions:

$$z\langle x\rangle \triangleright x\langle x\rangle \quad \Longrightarrow \quad \begin{cases} z^s\langle x, p, a, l\rangle \triangleright p\,\langle l, \emptyset, \{x\langle x\rangle\}\rangle \\ z^z\langle x, p, a, l\rangle \triangleright x\,\langle x, p, a, l\rangle \end{cases}$$

- The encoding function takes into account the scope of names: $[[\ \_\ ]]_{S,B}$



## Encoding: Merge Names

- **def** b⟨⟩ ▸… **in** [b⟨⟩ **:** 0] has two different behaviors:
  - It can commit
  - It can compute with the global reaction rule for b⟨⟩
- Merge definitions are encoded with two rules
  - One encoding the commit of the thread
  - The other, the application of the rule inside a negotiation
- Moreover, the commit behavior is allowed only when parameters are global names



## Correctness and Completeness

- Correctness
  P: $\Box_1$ and canonical. If P $\rightarrow_{cJ}*$ P′ with P: $\Box_1$, then ∃ Q s.t. $[[P]]_{fn(P),\emptyset} \rightarrow_j * Q$, and norm(Q) ≈ $[[P']]_{fn(P'),\emptyset}$
- Completeness
  P: $\Box_1$ and canonical. If $[[P]]_{fn(P),\emptyset} \rightarrow_j* Q$ and norm(Q) is well-defined, then ∃ P′ s.t. P $\rightarrow_{cJ}$ P′, and norm(Q) ≈ $[[P']]_{fn(P'),\emptyset}$



## Concluding remarks

- Flat cJoin can be implemented in Join
  - Commit is fully distributed
- This suggest that full cJoin can be modeled back in Join
  - At commit, a sub-negotiation can generate its parent:
    - new threads, and
    - messages to be delivered at commit
  - On abort, sub-negotiations should finish but not compensate
- Extension of running implementations of join (Jocaml, Comega, Join-Java)