

# A formalisation of the join-calculus in type theory

Jean-François Monin

Université Joseph Fourier & Verimag  
Grenoble, France

Work started at France Telecom R&D,  
with C. Bergerot

Work in progress, including slides

# Motivations

Practice of formal methods: 3 potential weaknesses

- ▶ relevance of the model
- ▶ complexity
- ▶ tedious parts

# Models for distributed computations

The underlying model should rely on assumptions

- ▶ realistic (close to the real thing)
- ▶ explicit

## Distributed systems

- ▶ calculi based on a **centralized** view of the world
  - ▷ CCS, CSP, (basic)  $\pi$ -calculus . . .
  - ▷ global state transition systems

⇒ distributed consensus
- ▶ calculi based on **local** interactions
  - ▷ join-calculus
  - ▷ proof theoretical approaches

⇒ local reasoning

# Complexity

Complete formalisation

⇒ everything is explicit

⇒ complexity tends to become unmanageable

- ▶ **basic design decisions** are fundamental:  
spent time on **simplicity** and **uniformity**
- ▶ **support tools** e.g. Coq, HOL, ...  
**tedious** parts must be checked

Useful features:

- ▶ higher order reasoning
- ▶ inductive reasoning
- ▶ reliability of support tools, kernel based (LCF)

# Outline

The join-calculus

Type theoretical concerns

Formalisation in CIC  
(Calculus of Inductive Constructions)

Research directions

# The Join Calculus

<http://pauillac.inria.fr/join/>

Fournet, Gonthier, Lévy, Maranget, Rémy

## A calculus of mobile processes, like $\pi$ -calculus

- ▶ **simple**: very small number of constructs
- ▶ less primitive than  $\pi$ -calculus in some sense...
- ▶ ...but **without hidden complexity**  
(s.t. distributed consensus, uncontrollable scopes)
  - ▷ model closer to real systems
  - ▷ efficient implementations: JC, [JoCaml](#)

## Several versions

- ▶ core, asynchronous / synchronous, pure / +constants

## Example

```
def mkcell⟨v0, κ0⟩ ▷  
  def set⟨u⟩, κ | cell⟨v⟩ ▷ cell⟨u⟩ | κ⟨⟩  
    & get⟨κ⟩ | cell⟨v⟩ ▷ cell⟨v⟩ | κ⟨v⟩  
  in cell⟨v0⟩ | κ0⟨get, set⟩  
in ⟨get1, set1⟩ := mkcell⟨0⟩ ; ... set1⟨3⟩ ; ... x := get1⟨⟩ ; ...
```

---

$s\langle u \rangle \mid c\langle v \rangle \triangleright c\langle u \rangle \mid \kappa\langle \rangle$

$g\langle \kappa \rangle \mid c\langle v \rangle \triangleright c\langle v \rangle \mid \kappa\langle v \rangle$

$M\langle v_0, \kappa_0 \rangle \triangleright$    

$s_1\langle u \rangle \mid c_1\langle v \rangle \triangleright c_1\langle u \rangle \mid \kappa\langle \rangle$

## Example

```
def mkcell⟨v0, κ0⟩ ▷  
  def set⟨u⟩, κ | cell⟨v⟩ ▷ cell⟨u⟩ | κ⟨⟩  
    & get⟨κ⟩ | cell⟨v⟩ ▷ cell⟨v⟩ | κ⟨v⟩  
  in cell⟨v0⟩ | κ0⟨get, set⟩  
in ⟨get1, set1⟩ := mkcell⟨0⟩ ; ... set1⟨3⟩ ; ... x := get1⟨⟩ ; ...
```

---

$s\langle u \rangle \mid c\langle v \rangle \triangleright c\langle u \rangle \mid \kappa\langle \rangle$

$g\langle \kappa \rangle \mid c\langle v \rangle \triangleright c\langle v \rangle \mid \kappa\langle v \rangle$

$M\langle v_0, \kappa_0 \rangle \triangleright$    

$s_1\langle u \rangle \mid c_1\langle v \rangle \triangleright c_1\langle u \rangle \mid \kappa\langle \rangle$

## Example

```
def mkcell⟨v0, κ0⟩ ▷  
  def set⟨u⟩, κ | cell⟨v⟩ ▷ cell⟨u⟩ | κ⟨⟩  
    & get⟨κ⟩ | cell⟨v⟩ ▷ cell⟨v⟩ | κ⟨v⟩  
  in cell⟨v0⟩ | κ0⟨get, set⟩  
in ⟨get1, set1⟩ := mkcell⟨0⟩ ; ... set1⟨3⟩ ; ... x := get1⟨⟩ ; ...
```

---

$s\langle u \rangle \mid c\langle v \rangle \triangleright c\langle u \rangle \mid \kappa\langle \rangle$

$g\langle \kappa \rangle \mid c\langle v \rangle \triangleright c\langle v \rangle \mid \kappa\langle v \rangle$

$M\langle v_0, \kappa_0 \rangle \triangleright$    

$s_1\langle u \rangle \mid c_1\langle v \rangle \triangleright c_1\langle u \rangle \mid \kappa\langle \rangle$

## Example

```
def mkcell⟨v0, κ0⟩ ▷  
  def set⟨u⟩, κ | cell⟨v⟩ ▷ cell⟨u⟩ | κ⟨⟩  
    & get⟨κ⟩ | cell⟨v⟩ ▷ cell⟨v⟩ | κ⟨v⟩  
  in cell⟨0⟩ | ... | get⟨x⟩ | ... | set⟨3⟩ | ... cell⟨v0⟩ | κ0⟨get, set⟩  
in ⟨get1, set1⟩ := mkcell⟨0⟩ ; ... set1⟨3⟩ ; ... x := get1⟨⟩ ; ...
```

---

$s\langle u \rangle \mid c\langle v \rangle \triangleright c\langle u \rangle \mid \kappa\langle \rangle$

$g\langle \kappa \rangle \mid c\langle v \rangle \triangleright c\langle v \rangle \mid \kappa\langle v \rangle$

$M\langle v_0, \kappa_0 \rangle \triangleright$    

$s_1\langle u \rangle \mid c_1\langle v \rangle \triangleright c_1\langle u \rangle \mid \kappa\langle \rangle$

## Example

```
def mkcell⟨v0, κ0⟩ ▷  
  def set⟨u, κ⟩ | cell⟨v⟩ ▷ cell⟨u⟩ | κ⟨⟩  
    & get⟨κ⟩ | cell⟨v⟩ ▷ cell⟨v⟩ | κ⟨v⟩  
  in cell⟨0⟩ | ... | get⟨x⟩ | ... | set⟨3, d⟩ | ... cell⟨v0⟩ | κ0⟨get, set⟩  
in ⟨get1, set1⟩ := mkcell⟨0⟩ ; ... set1⟨3⟩ ; ... x := get1⟨⟩ ; ...
```

---

$s\langle u, \kappa \rangle \mid c\langle v \rangle \triangleright c\langle u \rangle \mid \kappa\langle \rangle$   
 $g\langle \kappa \rangle \mid c\langle v \rangle \triangleright c\langle v \rangle \mid \kappa\langle v \rangle$

$M\langle v_0, \kappa_0 \rangle \triangleright$    

$s_1\langle u, \kappa \rangle \mid c_1\langle v \rangle \triangleright c_1\langle u \rangle \mid \kappa\langle \rangle$

## Example

```
def mkcell⟨v0, κ0⟩ ▷  
  def set⟨u, κ⟩ | cell⟨v⟩ ▷ cell⟨u⟩ | κ⟨⟩  
    & get⟨κ⟩ | cell⟨v⟩ ▷ cell⟨v⟩ | κ⟨v⟩  
  in cell⟨0⟩ | ... x := get⟨⟩ ; ... set⟨3⟩ ; ... cell⟨v0⟩ | κ0⟨get, set⟩  
in ⟨get1, set1⟩ := mkcell⟨0⟩ ; ... set1⟨3⟩ ; ... x := get1⟨⟩ ; ...
```

---

$s\langle u, \kappa \rangle \mid c\langle v \rangle \triangleright c\langle u \rangle \mid \kappa\langle \rangle$   
 $g\langle \kappa \rangle \mid c\langle v \rangle \triangleright c\langle v \rangle \mid \kappa\langle v \rangle$

$M\langle v_0, \kappa_0 \rangle \triangleright$    

$s_1\langle u, \kappa \rangle \mid c_1\langle v \rangle \triangleright c_1\langle u \rangle \mid \kappa\langle \rangle$

## Example

```
def mkcell⟨v0, κ0⟩ ▷  
  def set⟨u, κ⟩ | cell⟨v⟩ ▷ cell⟨u⟩ | κ⟨⟩  
    & get⟨κ⟩ | cell⟨v⟩ ▷ cell⟨v⟩ | κ⟨v⟩  
  in cell⟨v0⟩ | κ0⟨get, set⟩  
in ⟨get1, set1⟩ := mkcell⟨0⟩ ; ... set1⟨3⟩ ; ... x := get1⟨⟩ ; ...
```

---

$M\langle v_0, \kappa_0 \rangle \triangleright$    

$s_1\langle u, \kappa \rangle | c_1\langle v \rangle \triangleright c_1\langle u \rangle | \kappa\langle \rangle$   
 $g_1\langle \kappa \rangle | c_1\langle v \rangle \triangleright c_1\langle v \rangle | \kappa\langle v \rangle$

## Example

```
def mkcell⟨v0, κ0⟩ ▷  
  def set⟨u, κ⟩ | cell⟨v⟩ ▷ cell⟨u⟩ | κ⟨⟩  
    & get⟨κ⟩ | cell⟨v⟩ ▷ cell⟨v⟩ | κ⟨v⟩  
  in cell⟨v0⟩ | κ0⟨get, set⟩  
in ⟨get1, set1⟩ := mkcell⟨0⟩; ... set1⟨3⟩; ... x := get1⟨⟩; ... in ⟨get1, set1⟩
```

---

$M\langle v_0, \kappa_0 \rangle \triangleright$    

$s_1\langle u, \kappa \rangle | c_1\langle v \rangle \triangleright c_1\langle u \rangle | \kappa\langle \rangle$   
 $g_1\langle \kappa \rangle | c_1\langle v \rangle \triangleright c_1\langle v \rangle | \kappa\langle v \rangle$

## Example

```
def mkcell⟨v0, κ0⟩ ▷  
  def set⟨u, κ⟩ | cell⟨v⟩ ▷ cell⟨u⟩ | κ⟨⟩  
    & get⟨κ⟩ | cell⟨v⟩ ▷ cell⟨v⟩ | κ⟨v⟩  
  in cell⟨v0⟩ | κ0⟨get, set⟩  
in ⟨get1, set1⟩ := mkcell⟨0⟩ ; ... set1⟨3⟩ ; ... x := get1⟨⟩ ; ...
```

---

$M\langle v_0, \kappa_0 \rangle \triangleright$    

$M\langle 0, a \rangle$

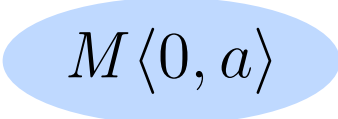
$s_1\langle u, \kappa \rangle | c_1\langle v \rangle \triangleright c_1\langle u \rangle | \kappa\langle \rangle$   
 $g_1\langle \kappa \rangle | c_1\langle v \rangle \triangleright c_1\langle v \rangle | \kappa\langle v \rangle$

## Example

```
def mkcell $\langle v_0, \kappa_0 \rangle \triangleright$   
  def set $\langle u, \kappa \rangle \mid \text{cell}\langle v \rangle \triangleright \text{cell}\langle u \rangle \mid \kappa\langle \rangle$   
    & get $\langle \kappa \rangle \mid \text{cell}\langle v \rangle \triangleright \text{cell}\langle v \rangle \mid \kappa\langle v \rangle$   
  in cell $\langle v_0 \rangle \mid \kappa_0\langle \text{get}, \text{set} \rangle$   
in  $\langle \text{get}_1, \text{set}_1 \rangle := \text{mkcell}\langle 0 \rangle ; \dots \text{set}_1\langle 3 \rangle ; \dots x := \text{get}_1\langle \rangle ; \dots$ 
```

---

$M\langle v_0, \kappa_0 \rangle \triangleright$    

  $M\langle 0, a \rangle$

$s_1\langle u, \kappa \rangle \mid c_1\langle v \rangle \triangleright c_1\langle u \rangle \mid \kappa\langle \rangle$

$g_1\langle \kappa \rangle \mid c_1\langle v \rangle \triangleright c_1\langle v \rangle \mid \kappa\langle v \rangle$

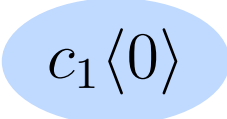
# Example

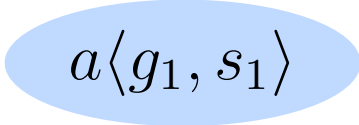
```
def mkcell⟨v0, κ0⟩ ▷  
  def set⟨u, κ⟩ | cell⟨v⟩ ▷ cell⟨u⟩ | κ⟨⟩  
    & get⟨κ⟩ | cell⟨v⟩ ▷ cell⟨v⟩ | κ⟨v⟩  
  in cell⟨v0⟩ | κ0⟨get, set⟩  
in ⟨get1, set1⟩ := mkcell⟨0⟩ ; ... set1⟨3⟩ ; ... x := get1⟨⟩ ; ...
```

---

$M\langle v_0, \kappa_0 \rangle \triangleright$    

$s_1\langle u, \kappa \rangle | c_1\langle v \rangle \triangleright c_1\langle u \rangle | \kappa\langle \rangle$   
 $g_1\langle \kappa \rangle | c_1\langle v \rangle \triangleright c_1\langle v \rangle | \kappa\langle v \rangle$

  
 $c_1\langle 0 \rangle$

  
 $a\langle g_1, s_1 \rangle$

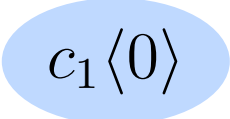
## Example

```
def mkcell⟨v0, κ0⟩ ▷  
  def set⟨u, κ⟩ | cell⟨v⟩ ▷ cell⟨u⟩ | κ⟨⟩  
    & get⟨κ⟩ | cell⟨v⟩ ▷ cell⟨v⟩ | κ⟨v⟩  
  in cell⟨v0⟩ | κ0⟨get, set⟩  
in ⟨get1, set1⟩ := mkcell⟨0⟩ ; ... set1⟨3⟩ ; ... x := get1⟨⟩ ; ...
```

---

$M\langle v_0, \kappa_0 \rangle \triangleright$    

$s_1\langle u, \kappa \rangle | c_1\langle v \rangle \triangleright c_1\langle u \rangle | \kappa\langle \rangle$   
 $g_1\langle \kappa \rangle | c_1\langle v \rangle \triangleright c_1\langle v \rangle | \kappa\langle v \rangle$

  
 $c_1\langle 0 \rangle$

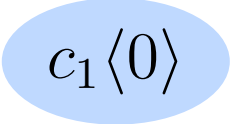
## Example

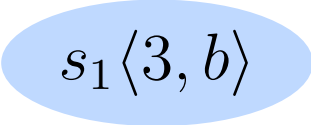
```
def mkcell⟨v0, κ0⟩ ▷  
  def set⟨u, κ⟩ | cell⟨v⟩ ▷ cell⟨u⟩ | κ⟨⟩  
    & get⟨κ⟩ | cell⟨v⟩ ▷ cell⟨v⟩ | κ⟨v⟩  
  in cell⟨v0⟩ | κ0⟨get, set⟩  
in ⟨get1, set1⟩ := mkcell⟨0⟩ ; ... set1⟨3⟩ ; ... x := get1⟨⟩ ; ...
```

---

$M\langle v_0, \kappa_0 \rangle \triangleright$    

$s_1\langle u, \kappa \rangle | c_1\langle v \rangle \triangleright c_1\langle u \rangle | \kappa\langle \rangle$   
 $g_1\langle \kappa \rangle | c_1\langle v \rangle \triangleright c_1\langle v \rangle | \kappa\langle v \rangle$

 $c_1\langle 0 \rangle$

 $s_1\langle 3, b \rangle$

## Example

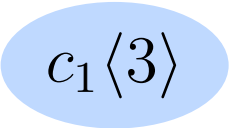
```
def mkcell⟨v0, κ0⟩ ▷  
  def set⟨u, κ⟩ | cell⟨v⟩ ▷ cell⟨u⟩ | κ⟨⟩  
    & get⟨κ⟩ | cell⟨v⟩ ▷ cell⟨v⟩ | κ⟨v⟩  
  in cell⟨v0⟩ | κ0⟨get, set⟩  
in ⟨get1, set1⟩ := mkcell⟨0⟩ ; ... set1⟨3⟩ ; ... x := get1⟨⟩ ; ...
```

---

$M\langle v_0, \kappa_0 \rangle \triangleright$    

$s_1\langle u, \kappa \rangle | c_1\langle v \rangle \triangleright c_1\langle u \rangle | \kappa\langle \rangle$   
 $g_1\langle \kappa \rangle | c_1\langle v \rangle \triangleright c_1\langle v \rangle | \kappa\langle v \rangle$

  $b\langle \rangle$

  $c_1\langle 3 \rangle$

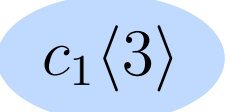
## Example

```
def mkcell⟨v0, κ0⟩ ▷  
  def set⟨u, κ⟩ | cell⟨v⟩ ▷ cell⟨u⟩ | κ⟨⟩  
    & get⟨κ⟩ | cell⟨v⟩ ▷ cell⟨v⟩ | κ⟨v⟩  
  in cell⟨v0⟩ | κ0⟨get, set⟩  
in ⟨get1, set1⟩ := mkcell⟨0⟩ ; ... set1⟨3⟩ ; ... x := get1⟨⟩ ; ...
```

---

$M\langle v_0, \kappa_0 \rangle \triangleright$    

$s_1\langle u, \kappa \rangle | c_1\langle v \rangle \triangleright c_1\langle u \rangle | \kappa\langle \rangle$   
 $g_1\langle \kappa \rangle | c_1\langle v \rangle \triangleright c_1\langle v \rangle | \kappa\langle v \rangle$

  
 $c_1\langle 3 \rangle$

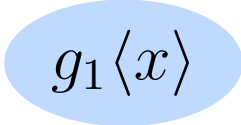
## Example

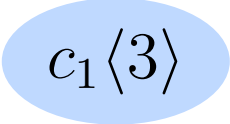
```
def mkcell⟨v0, κ0⟩ ▷  
  def set⟨u, κ⟩ | cell⟨v⟩ ▷ cell⟨u⟩ | κ⟨⟩  
    & get⟨κ⟩ | cell⟨v⟩ ▷ cell⟨v⟩ | κ⟨v⟩  
  in cell⟨v0⟩ | κ0⟨get, set⟩  
in ⟨get1, set1⟩ := mkcell⟨0⟩ ; ... set1⟨3⟩ ; ... x := get1⟨⟩ ; ...
```

---

$M\langle v_0, \kappa_0 \rangle \triangleright$    

$s_1\langle u, \kappa \rangle | c_1\langle v \rangle \triangleright c_1\langle u \rangle | \kappa\langle \rangle$   
 $g_1\langle \kappa \rangle | c_1\langle v \rangle \triangleright c_1\langle v \rangle | \kappa\langle v \rangle$

  
 $g_1\langle x \rangle$

  
 $c_1\langle 3 \rangle$

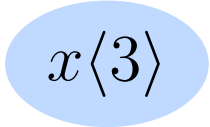
## Example

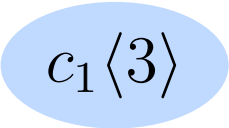
```
def mkcell⟨v0, κ0⟩ ▷  
  def set⟨u, κ⟩ | cell⟨v⟩ ▷ cell⟨u⟩ | κ⟨⟩  
    & get⟨κ⟩ | cell⟨v⟩ ▷ cell⟨v⟩ | κ⟨v⟩  
  in cell⟨v0⟩ | κ0⟨get, set⟩  
in ⟨get1, set1⟩ := mkcell⟨0⟩ ; ... set1⟨3⟩ ; ... x := get1⟨⟩ ; ...
```

---

$M\langle v_0, \kappa_0 \rangle \triangleright$    

$s_1\langle u, \kappa \rangle | c_1\langle v \rangle \triangleright c_1\langle u \rangle | \kappa\langle \rangle$   
 $g_1\langle \kappa \rangle | c_1\langle v \rangle \triangleright c_1\langle v \rangle | \kappa\langle v \rangle$

 $x\langle 3 \rangle$

 $c_1\langle 3 \rangle$

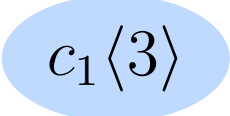
## Example

```
def mkcell⟨v0, κ0⟩ ▷  
  def set⟨u, κ⟩ | cell⟨v⟩ ▷ cell⟨u⟩ | κ⟨⟩  
    & get⟨κ⟩ | cell⟨v⟩ ▷ cell⟨v⟩ | κ⟨v⟩  
  in cell⟨v0⟩ | κ0⟨get, set⟩  
in ⟨get1, set1⟩ := mkcell⟨0⟩ ; ... set1⟨3⟩ ; ... x := get1⟨⟩ ; ...
```

---

$M\langle v_0, \kappa_0 \rangle \triangleright$    

$s_1\langle u, \kappa \rangle | c_1\langle v \rangle \triangleright c_1\langle u \rangle | \kappa\langle \rangle$   
 $g_1\langle \kappa \rangle | c_1\langle v \rangle \triangleright c_1\langle v \rangle | \kappa\langle v \rangle$

  
 $c_1\langle 3 \rangle$

## Official formal definition

$P \stackrel{\text{def}}{=} x\langle \overline{a} \rangle \quad | \quad \text{def } D \text{ in } P \quad | \quad P \mid P \quad | \quad \mathbf{0}$

$D \stackrel{\text{def}}{=} J \triangleright P \quad | \quad D \& D \quad | \quad \mathbf{T}$

$J \stackrel{\text{def}}{=} x\langle \overline{v} \rangle \quad | \quad J \mid J$

**def**  $mkcell\langle v_0, \kappa_0 \rangle \triangleright$

**def**  $set\langle u, \kappa \rangle \mid cell\langle v \rangle \triangleright cell\langle u \rangle \mid \kappa\langle \rangle$

$\& get\langle \kappa \rangle \mid cell\langle v \rangle \triangleright cell\langle v \rangle \mid \kappa\langle v \rangle$

**in**  $cell\langle v_0 \rangle \mid \kappa_0\langle get, set \rangle$

**in**  $\dots mkcell\langle 0, a \rangle \dots set_1\langle 3, b \rangle \dots x := get_1\langle \rangle \dots$

# Reflexive chemical abstract machines

$$\begin{array}{l}
 P \stackrel{\text{def}}{=} x\langle \vec{a} \rangle \quad | \quad \text{def } D \text{ in } P \quad | \quad P \mid P \quad | \quad \mathbf{0} \\
 D \stackrel{\text{def}}{=} J \triangleright P \quad | \quad D \& D \quad | \quad \mathbf{T} \\
 J \stackrel{\text{def}}{=} x\langle \vec{v} \rangle \quad | \quad J \mid J
 \end{array}$$

multiset of *Reac*  $\vdash$  multiset of *Proc*

$$\begin{array}{l}
 \overrightarrow{x\langle \vec{v} \rangle} \triangleright S \vdash \overrightarrow{x\langle \vec{a} \rangle} \quad \rightarrow \quad \overrightarrow{x\langle \vec{v} \rangle} \triangleright S \vdash S[\vec{v} := \vec{a}] \\
 \vdash \text{def } D \text{ in } S \quad \Leftrightarrow \quad D_\nu \vdash S_\nu
 \end{array}$$

structural rules on  $|$  and  $\&$


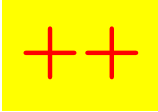
**Distributed RCHAM:** multiset of located RCHAM

# Type Theory

**Theory:** formalize mathematics with

- ▶ typed  $\lambda$ -calculus
- ▶ Inductive types
  - ▷ related proof and computation principles
- ▶ Proof theoretic constructs
  - ▷ proof = program

## Practice

- ▶ safe proof assistants  Coq, LEGO, Agda
- ▶ tree-like structures 

# Example

$0 : \text{nat}$        $S : \text{nat} \rightarrow \text{nat}$

$\text{plus} : \text{nat} \rightarrow \text{nat} \rightarrow \text{nat}$

$$\frac{P : \text{nat} \rightarrow \text{Prop} \quad P\ 0 \quad \forall n : \text{nat}, P\ n \rightarrow P(S\ n)}{\forall n : \text{nat}, P\ n} \text{nat\_ind}$$

$$\frac{}{\text{nat}} 0 \quad \frac{\text{nat}}{\text{nat}} S \quad \frac{\text{nat} \quad \text{nat}}{\text{nat}} \text{plus}$$

$$\frac{\frac{\frac{}{\text{nat}} 0 \quad \frac{\frac{}{\text{nat}} 0}{\text{nat}} S}{\text{nat}} \text{plus}}{\text{nat}} \text{plus} \quad \frac{\frac{\frac{}{\text{nat}} 0 \quad \frac{}{\text{nat}} 0}{\text{nat}} \text{plus}}{\text{nat}} S}{\text{nat}} \text{plus}}{\text{nat}} \text{plus}$$

# Issues

## **Names and scoping rules** (*does not depend on TT*)

- ▶ (De Bruijn) <sup>$n+p$</sup>
- ▶ active names

## **Multisets**

- ▶ lists + take
- ▶ space

## **Mutual inductives**

- ▶ replaced with polymorphic constructs + extensions by morphism

## Structural rules $\rightarrow$ multisets

$$P \stackrel{\text{def}}{=} x\langle\vec{a}\rangle \quad | \quad \text{def } D \text{ in } P \quad | \quad P \mid P \quad | \quad \mathbf{0}$$

$$D \stackrel{\text{def}}{=} J \triangleright P \quad | \quad D \& D \quad | \quad \mathbf{T}$$

$$J \stackrel{\text{def}}{=} x\langle\vec{v}\rangle \quad | \quad J \mid J$$

$$Proc \stackrel{\text{def}}{=} x\langle\vec{a}\rangle \quad | \quad \text{def } Def \text{ in } Sol$$

$$Sol \stackrel{\text{def}}{=} \text{multiset of } Proc$$

$$Def \stackrel{\text{def}}{=} \text{multiset of } Reac$$

$$Reac \stackrel{\text{def}}{=} \text{multiset of } x\langle\vec{v}\rangle \quad \triangleright \quad Sol$$

# Multisets = lists + take

$$\frac{}{\forall l : \text{list } A, \text{take } A \ x \ x :: l \ l} \text{take\_hd}$$

$$\frac{\forall l_1 \ l_2 : \text{list } A, \forall y : A, \text{take } A \ x \ l_1 \ l_2}{\text{take } A \ x \ y :: l_1 \ y :: l_2} \text{take\_tl}$$

$\left. \begin{array}{l} \text{take}^+ \\ \text{take}^* \end{array} \right\} \textit{standard}$

## Structural rules $\rightarrow$ lists

$P \stackrel{\text{def}}{=} x\langle\vec{a}\rangle \mid \text{def } D \text{ in } P \mid P \mid P \mid \mathbf{0}$

$D \stackrel{\text{def}}{=} J \triangleright P \mid D \& D \mid \mathbf{T}$

$J \stackrel{\text{def}}{=} x\langle\vec{v}\rangle \mid J \mid J$

$Proc \stackrel{\text{def}}{=} x\langle\vec{a}\rangle \mid \text{def } Def \text{ in } Sol$

$Sol \stackrel{\text{def}}{=} \text{list } Proc$

$Def \stackrel{\text{def}}{=} \text{list } Reac$

$Reac \stackrel{\text{def}}{=} \text{list } x\langle\vec{v}\rangle \triangleright Sol$

## Folding mutual inductives

$Proc \stackrel{\text{def}}{=} x\langle \overrightarrow{a} \rangle \quad | \quad \text{def } Def \text{ in } Sol$

$Sol \stackrel{\text{def}}{=} \text{list } Proc$

$Def \stackrel{\text{def}}{=} \text{list } Reac$

$Reac \stackrel{\text{def}}{=} \text{list } x\langle \overrightarrow{v} \rangle \triangleright Sol$

The first definition becomes

$Proc \stackrel{\text{def}}{=} x\langle \overrightarrow{a} \rangle$   
 $| \quad \text{def } (\text{list } (\text{list } x\langle \overrightarrow{v} \rangle \triangleright \text{list } Proc) \text{ in } \text{list } Proc$

The others are no longer recursive

Then use polymorphic combinators

## Folding mutual inductives

```
Fixpoint proc_valid_at_depth (nc np: nat) (p: process)
  {struct p}: Prop :=
  match p with
  | PMessage m => message_valid_at_depth nc np m
  | PDef ar lr lp =>
    fold_lreact (proc_valid_at_depth (S nc) (S np)) lr ^
    fold_and (proc_valid_at_depth (S nc) np) lp
  end.
```

```
Definition sol_valid_at_depth (nc np: nat) (lp: solution): Prop :=
  fold_and (proc_valid_at_depth nc np) lp.
```

```
Definition lreact_valid_at_depth (nc np: nat) (lr: list react): Prop :=
  fold_lreact (proc_valid_at_depth nc np) lr.
```

# Names of active channels

Have to be distinguished from names of nested channels

- ▶ channels used as actual parameters of active messages must be active

Representing deterministic behaviors by non deterministic choices is bad

- ▶ loss of information
- ▶ technically: *inversions* (much) more complicated

⇒

- ▶ related reactions are grouped together within an automaton (as in the implementation)
- ▶ automata are given a unique name
- ▶ active channel name = ⟨ automaton name, rank ⟩

# What is the name of an automaton?

named automata  $\vdash$  list *Proc*

name $\rightarrow$ automaton  $\vdash$  list *Def*, list *SMsg*

automaton : list *RMsg*  $\times$  list *AReac*  $\times$  arity

*RMsg* : rank  $\times$  actual parameters

*SMsg* : name  $\times$  rank  $\times$  actual parameters

**It's just a (logical) location!**

location $\rightarrow$ automaton  $\vdash$  list *Def*, list *SMsg*

distribution = mapping

logical location  $\rightarrow$  physical location

# Scopes

# Spatial RCHAM

location  $\rightarrow$  automaton  $\vdash$  list *Def*, list *SMsg*

*Proc*  $\stackrel{\text{def}}{=}$   $x\langle\vec{a}\rangle$  | **def** list list ... **in** list *Proc*

*Sol*  $\stackrel{\text{def}}{=}$  list *Proc*

*Reac*  $\stackrel{\text{def}}{=}$  list  $x\langle\vec{v}\rangle \triangleright$  *Sol*

*Def*  $\stackrel{\text{def}}{=}$  list *Reac*

$$\overrightarrow{x\langle\vec{a}\rangle}, \overrightarrow{x\langle\vec{v}\rangle} \triangleright S \vdash \longrightarrow \overrightarrow{x\langle\vec{v}\rangle} \triangleright S \vdash S[\vec{v} := \vec{a}]$$

$$\vdash \mathbf{def} D \mathbf{in} S \longrightarrow \emptyset, D_\nu \vdash \mathcal{D}S_\nu, \mathcal{M}S_\nu$$

$$l \vdash \langle p, r, a \rangle \longrightarrow l[l p := l p.\langle r, a \rangle] \vdash$$

# Achievements & provocative conclusion

Formal definition completely checked in Coq

(< 1000 lines)

De Bruijn indices correctly handled:

- ▶ sent messages make sense
- ▶ two separate kinds of names: active and nested

A better definition of join-calculus:

- ▶ explicit localities: distribution for free
- ▶ no structural rules, no reversible rules, no congruences
- ▶ explicit message migration
- ▶ conjecture : reversibility of str-def actually does not scale up to DRCHAM in the original model