On the Introduction of Guarded Lists in Bach: Expressiveness, Correctness, and Efficiency Issues

Manel Barkallah – Jean-Marie Jacquet

Namur Digital Research Institute
University of Namur, Belgium

June 2023
Coordination as a powerful paradigm

- a concurrent framework based on shared information

- clear separation between interactional and computational aspects
  - many models and languages
  - many theoretical pieces of work
  - many implementations
Coordination as a powerful paradigm . . . but in practice

- a concurrent framework based on shared information

- clear separation between interactional and computational aspects
  - many models and languages
  - many theoretical pieces of work
  - many implementations

- in practice, how to construct programs?
  - how to describe (real-life) problems?
  - how to reason on the programs?
  - how to be sure that what is described by the programs corresponds to what has to be modelled?
Coordination as a powerful paradigm . . . but in practice

- a concurrent framework based on shared information

- clear separation between interactional and computational aspects
  - many models and languages
  - many theoretical pieces of work
  - many implementations

- in practice, how to construct programs?
  - how to describe (real-life) problems?
  - how to reason on the programs?
  - how to be sure that what is described by the programs corresponds to what has to be modelled?
Coordination as a powerful paradigm... but in practice

- A concurrent framework based on shared information
- Clear separation between interactional and computational aspects
  - Many models and languages
  - Many theoretical pieces of work
  - Many implementations

In practice, how to construct programs?
- How to describe (real-life) problems?
- How to reason on the programs?
- How to be sure that what is described by the programs corresponds to what has to be modelled?
Coordination as a powerful paradigm . . . but in practice

- a concurrent framework based on shared information

- clear separation between interactional and computational aspects
  - many models and languages
  - many theoretical pieces of work
  - many implementations

- in practice, how to construct programs?
  - how to describe (real-life) problems?
  - how to reason on the programs?
  - how to be sure that what is described by the programs corresponds to what has to be modelled?
Coordination as a powerful paradigm ... but in practice

- a concurrent framework based on shared information

- clear separation between interactional and computational aspects
  - many models and languages
  - many theoretical pieces of work
  - many implementations

- in practice, how to construct programs?
  - how to describe (real-life) problems?
  - how to reason on the programs?
  - how to be sure that what is described by the programs corresponds to what has to be modelled?
Coordination as a powerful paradigm... but in practice

- a concurrent framework based on shared information
- clear separation between interactional and computational aspects
  - many models and languages
  - many theoretical pieces of work
  - many implementations

- in practice, how to construct programs?
  - how to describe (real-life) problems?
  - how to reason on the programs?
  - how to be sure that what is described by the programs corresponds to what has to be modelled?
Coordination as a powerful paradigm . . . but in practice

- a concurrent framework based on shared information

- clear separation between interactional and computational aspects
  - many models and languages
  - many theoretical pieces of work
  - many implementations

- in practice, how to construct programs?
  - how to describe (real-life) problems?
  - how to reason on the programs?
  - how to be sure that what is described by the programs corresponds to what has to be modelled?
Coordination as a powerful paradigm . . . but in practice

- a concurrent framework based on shared information
- clear separation between interactional and computational aspects
  - many models and languages
  - many theoretical pieces of work
  - many implementations
- in practice, how to construct programs?
  - how to describe (real-life) problems?
  - how to reason on the programs?
  - how to be sure that what is described by the programs corresponds to what has to be modelled?
\( \text{tell}(t) \)
$tell(t)$
The Bach Coordination Language

$\text{tell}(t)$  

$\text{ask}(t)$
The Bach Coordination Language

tell(t)

ask(t)

get(t)

$\textit{t}$
The Bach Coordination Language

tell\(t\)  
get\(t\)  
ask\(t\)
The Bach Coordination Language

tell(t) → ask(t)

nask(t) ← get(t)
Transition system

(T) \[ \langle \text{tell}(t) \mid \sigma \rangle \rightarrow \langle E \mid \sigma \cup \{t\} \rangle \]

(A) \[ \langle \text{ask}(t) \mid \sigma \cup \{t\} \rangle \rightarrow \langle E \mid \sigma \cup \{t\} \rangle \]

(G) \[ \langle \text{get}(t) \mid \sigma \cup \{t\} \rangle \rightarrow \langle E \mid \sigma \rangle \]

(N) \[ \frac{t \notin \sigma}{\langle \text{nask}(t) \mid \sigma \rangle \rightarrow \langle E \mid \sigma \rangle} \]
Rush hour as a running example
Modeling rush-hour

- trucks and cars as concurrent agents
- competing through free places on the shared space
- trucks and cars as concurrent agents
- competing through free places on the shared space
finite sets

\[
eset \text{RCInt} = \{ 1, 2, 3, 4, 5, 6 \}.
\]

maps and equations as rewriting rules

\[
\text{map down\_truck} : \text{RCInt} \rightarrow \text{RCInt}.
\]
\[
\text{eqn} \downtruck(1) = 4. \downtruck(2) = 5. \downtruck(3) = 6.
\]

structured pieces of information

- flat tokens: a, b, ..., t, u, ...
- composed terms: \( f(a_1, \ldots, a_n) \)
- free places as free(i, j)
finite sets

\[
eset \text{RCInt} = \{1, 2, 3, 4, 5, 6\}.
\]

maps and equations as rewriting rules

\[
\begin{align*}
\text{map} \downarrow \text{truck} : \text{RCInt} & \rightarrow \text{RCInt}. \\
\text{eqn} \downarrow \text{truck}(1) & = 4. \downarrow \text{truck}(2) = 5. \\
& \downarrow \text{truck}(3) = 6.
\end{align*}
\]

structured pieces of information

- flat tokens: a, b, ..., t, u, ...
- composed terms: \( f(a_1, \ldots, a_n) \)
- free places as \( \text{free}(i, j) \)
Data

- finite sets
  
  \[ \text{eset } \text{RCInt} = \{ 1, 2, 3, 4, 5, 6 \} \]

- maps and equations as rewriting rules

  \[
  \text{map} \quad \text{down\_truck} : \text{RCInt} \rightarrow \text{RCInt}.
  
  \text{eqn} \quad \text{down\_truck}(1) = 4. \quad \text{down\_truck}(2) = 5. \quad \text{down\_truck}(3) = 6.
  \]

- structured pieces of information
  
  - flat tokens: a, b, ..., t, u, ...
  - composed terms: \( f(a_1, \ldots, a_n) \)
  - free places as \( \text{free}(1, j) \)
finite sets

\[ \text{eset } \text{RCInt} = \{1, 2, 3, 4, 5, 6\} \]

maps and equations as rewriting rules

\[
\begin{align*}
\text{map } \text{down\_truck} : \text{RCInt} & \to \text{RCInt}. \\
\text{eqn } \text{down\_truck}(1) & = 4. \quad \text{down\_truck}(2) = 5. \\
& \quad \text{down\_truck}(3) = 6.
\end{align*}
\]

structured pieces of information

- flat tokens: \(a, b, \ldots, t, u, \ldots\)
- composed terms: \(f(a_1, \ldots, a_n)\)
- free places as \(\text{free}(i, j)\)
finite sets

\( \text{eset } \text{RCInt} = \{1, 2, 3, 4, 5, 6\} \).

maps and equations as rewriting rules

\textbf{map} down\_truck : \text{RCInt} \rightarrow \text{RCInt}.
\textbf{eqn} down\_truck(1) = 4. down\_truck(2) = 5.
\quad down\_truck(3) = 6.

structured pieces of information

- flat tokens: \(a, b, \ldots, t, u, \ldots\)
- composed terms: \(f(a_1, \ldots, a_n)\)
  - free places as \(\text{free}(i, j)\)
Data

- finite sets
  
  \[
  \text{eset RCInt} = \{1, 2, 3, 4, 5, 6\}.
  \]

- maps and equations as rewriting rules
  
  \[
  \text{map down_truck} : \text{RCInt} \to \text{RCInt}.
  \]
  \[
  \text{eqn down_truck}(1) = 4. \text{ down_truck}(2) = 5. \text{ down_truck}(3) = 6.
  \]

- structured pieces of information
  
  - flat tokens: a, b, \ldots, t, u, \ldots
  
  - composed terms: \(f(a_1, \ldots, a_n)\)
  
  - free places as free(i,j)
Data

- finite sets
  \[ \text{eset } \text{RCInt} = \{1, 2, 3, 4, 5, 6\}. \]

- maps and equations as rewriting rules
  \[
  \text{map } \text{down\_truck} : \text{RCInt} \rightarrow \text{RCInt}.
  \]
  \[
  \text{eqn } \text{down\_truck}(1) = 4. \text{down\_truck}(2) = 5.
  \text{down\_truck}(3) = 6.
  \]

- structured pieces of information
  - flat tokens: a, b, \ldots, t, u, \ldots
  - composed terms: \( f(a_1, \ldots, a_n) \)
  \[ \text{free places as free}(i, j) \]
finite sets

\[ \text{eset RCInt} = \{1, 2, 3, 4, 5, 6\} \]

maps and equations as rewriting rules

\[
\text{map down_truck : RCInt} \rightarrow \text{RCInt}.
\text{eqn down_truck}(1) = 4. \text{down_truck}(2) = 5. \text{down_truck}(3) = 6.
\]

structured pieces of information

- flat tokens: a, b, \ldots, t, u, \ldots
- composed terms: \( f(a_1, \ldots, a_n) \)
- free places as \( \text{free}(i, j) \)
Agents

\[ A ::= \text{Prim} \mid \text{Proc} \mid \]
\[ A ; A \mid A \parallel A \mid A + A \mid \]
\[ C \rightarrow A \diamond A \mid \sum_{e \in S} A_e \]
Agents

\[ A ::= \text{Prim} \mid \text{Proc} \mid \]

\[ A ; A \mid A \parallel A \mid A + A \mid \]

\[ C \rightarrow A \diamond A \mid \sum_{e \in S} A_e \]

where \textit{Prim} represents a primitive, \textit{Proc} a procedure call, \textit{C} a condition, \textit{e} a variable and \textit{S} a set.
Rush-hour with animations

\[
\text{eset } \text{RCInt} = \{ 1, 2, 3, 4, 5, 6 \}.
\]

\[
\text{Colors} = \{ \text{yellow, green, blue, purple, red, orange} \}.
\]

\[
\text{proc } \text{VerticalTruck}(r: \text{RCInt}, c: \text{RCInt}, p: \text{Colors}) =
\]

\[
( (r > 1 \& r < 5) \rightarrow \ ( \text{get(free(pred(r),c))});
\text{moveTruck(pred(r),c,p)};
\text{tell(free(succ(succ(r)),c))};
\text{VerticalTruck(pred(r),c,p)) })
\]

\[
+ ( (r < 4) \rightarrow \ ( \text{get(free(down_truck(r),c))});
\text{moveTruck(succ(r),c,p)};
\text{tell(free(r,c))};
\text{VerticalTruck(succ(r),c,p))}).
\]
• Key information on the store: $\#free(1, 1)$

• Basic formulae: equalities or inequalities involving integers and key information

  $\#free(1, 1) = 3$

• Propositional state formulae: combination of basic formulae by usual Boolean operators

• Linear temporal logic fragment:

  $TF ::= PF \mid \text{Next } TF \mid PF \text{ Until } TF$

• Reach formulae:

  $Reach(\#out = 1) \equiv \text{true Until } (\#out = 1)$
Key information on the store: \( \#free(1, 1) \)

Basic formulae: equalities or inequalities involving integers and key information

\[ \#free(1, 1) = 3 \]

Propositional state formulae: combination of basic formulae by usual Boolean operators

Linear temporal logic fragment:

\[ TF ::= PF \mid \text{Next } TF \mid PF \text{ Until } TF \]

Reach formulae:

\[ \text{Reach} (\#out = 1) \equiv \text{true Until} (\#out = 1) \]
Model checking

- Key information on the store: $\#\text{free}(1, 1)$

- Basic formulae: equalities or inequalities involving integers and key information
  - $\#\text{free}(1, 1) = 3$

- Propositional state formulae: combination of basic formulae by usual Boolean operators

- Linear temporal logic fragment:
  - $TF ::= PF \mid \text{Next } TF \mid PF \text{ Until } TF$

- Reach formulae:
  - $\text{Reach}(\#\text{out} = 1) \equiv \text{true Until } (\#\text{out} = 1)$
Key information on the store: $\#\text{free}(1, 1)$

Basic formulae: equalities or inequalities involving integers and key information

$\#\text{free}(1, 1) = 3$

Propositional state formulae: combination of basic formulae by usual Boolean operators

Linear temporal logic fragment:

$$TF ::= PF \mid \text{Next } TF \mid PF \text{ Until } TF$$

Reach formulae:

$$\text{Reach}(\#\text{out} = 1) \equiv \text{true Until } (\#\text{out} = 1)$$
Model checking

- Key information on the store: \( \#free(1, 1) \)

- Basic formulae: equalities or inequalities involving integers and key information
  \[ \#free(1, 1) = 3 \]

- Propositional state formulae: combination of basic formulae by usual Boolean operators

- Linear temporal logic fragment:
  \[ TF ::= Pf | \text{Next } TF | Pf \text{ Until } TF \]

- Reach formulae:
  \[ \text{Reach}(\#\text{out} = 1) \equiv \text{true Until}(\#\text{out} = 1) \]
Key information on the store: \( \#\text{free}(1, 1) \)

Basic formulae: equalities or inequalities involving integers and key information

\[ \#\text{free}(1, 1) = 3 \]

Propositional state formulae: combination of basic formulae by usual Boolean operators

Linear temporal logic fragment:

\[ TF ::= PF \mid \text{Next } TF \mid PF \text{ Until } TF \]

Reach formulae:

\[ \text{Reach}(\#\text{out} = 1) \equiv \text{true Until } (\#\text{out} = 1) \]
Key information on the store: $\#_{\text{free}}(1, 1)$

Basic formulae: equalities or inequalities involving integers and key information

$\#_{\text{free}}(1, 1) = 3$

Propositional state formulae: combination of basic formulae by usual Boolean operators

Linear temporal logic fragment:

$$TF ::= PF \mid \text{Next } TF \mid PF \text{ Until } TF$$

Reach formulae:

$$\text{Reach}(\#_{\text{out}} = 1) \equiv \text{true Until } (\#_{\text{out}} = 1)$$
Key information on the store: \( \#free(1, 1) \)

Basic formulae: equalities or inequalities involving integers and key information
\[ \#free(1, 1) = 3 \]

Propositional state formulae: combination of basic formulae by usual Boolean operators

Linear temporal logic fragment:
\[
TF ::= PF \mid \text{Next } TF \mid PF \text{ Until } TF
\]

Reach formulae:
\[
Reach(\#out = 1) \equiv \text{true Until } (\#out = 1)
\]
Current store

\[ a[1] \]

token: \[ \text{[input field]} \]
multiplicity: \[ 1 \]

Tell Get

New Autonomous Agent New Interactive Agent New Description New Model Checker
get(free(pred(r), c));
move(truck_img(c), pred(r), c);
tell(free(succ(succ(r)), c))

get(free(pred(r), c));
move(truck_img(c), pred(r), c);
tell(free(succ(succ(r)), c))
get(free(pred(r),c));
move(truck_img(c),pred(r),c);
tell(free(succ(succ(r)),c))
get(free(pred(r),c));
move(truck_img(c),pred(r),c);
tell(free(succ(succ(r)),c))

[ get(free(pred(r),c)) →
  move(truck_img(c),pred(r),c),
  tell(free(succ(succ(r)),c)) ]
A guarded list construct

The construct

\[ [p \rightarrow p_1, \ldots, p_n] \] where \( p, p_1, \ldots, p_n \) are primitives

\[
\begin{align*}
\text{(Le)} & \quad \langle [] | \sigma \rangle \longrightarrow \langle E | \sigma \rangle \\
\text{(Ln)} & \quad \langle p | \sigma \rangle \longrightarrow \langle E | \tau \rangle, \quad \langle L | \tau \rangle \longrightarrow^* \langle E | \phi \rangle \\
& \quad \langle [p|L] | \sigma \rangle \longrightarrow \langle E | \phi \rangle \\
\text{(GL)} & \quad \langle p | \sigma \rangle \longrightarrow \langle E | \tau \rangle, \quad \langle L | \tau \rangle \longrightarrow^* \langle E | \phi \rangle \\
& \quad \langle [p \rightarrow L] | \sigma \rangle \longrightarrow \langle E | \phi \rangle
\end{align*}
\]
Objectives

- Introduce a new construct called guarded list
- Establish an increase of expressiveness
- Propose a theory of refinement
- Show an increase of performance
Objectives

- Introduce a new construct called guarded list
- Establish an increase of expressiveness
- Propose a theory of refinement
- Show an increase of performance
Objectives

- Introduce a new construct called guarded list
- Establish an increase of expressiveness
- Propose a theory of refinement
- Show an increase of performance
Objectives

- Introduce a new construct called guarded list
- Establish an increase of expressiveness
- Propose a theory of refinement
- Show an increase of performance
Expressiveness

$L'$ embeds $L$
Propositions

- $\mathcal{L}_g(\text{ask, tell}) \not\subseteq \mathcal{L}_r(\text{ask, tell})$
- Inability for $\mathcal{L}_r(\text{ask, tell})$ to atomically test the presence of two distinct tokens $a$ and $b$.

- Assume $AB = [\text{ask}(a) \to \text{ask}(b)]$ and $C(AB)$ a coder (in $\mathcal{L}_r(\text{ask, tell})$)
- $C(AB)$ in general form:

$$
tell(t_1) ; A_1 + \cdots + tell(t_p) ; A_p
+ \text{ask}(u_1) ; B_1 + \cdots + \text{ask}(u_q) ; B_q
+ gp_1 ; C_1 + \cdots + gp_r ; C_r
$$
Propositions

- $\mathcal{L}_g(ask, tell) \not\subseteq \mathcal{L}_r(ask, tell)$
  - Inability for $\mathcal{L}_r(ask, tell)$ to atomically test the presence of two distinct tokens $a$ and $b$.

- Assume $AB = [ask(a) \rightarrow ask(b)]$ and $C(AB)$ a coder (in $\mathcal{L}_r(ask, tell)$)
- $C(AB)$ in general form:

  $$tell(t_1) ; A_1 + \cdots + tell(t_p) ; A_p$$
  $$+ ask(u_1) ; B_1 + \cdots + ask(u_q) ; B_q$$
  $$+ gp_1 ; C_1 + \cdots + gp_r ; C_r$$
### Expressiveness - Proof example

**Propositions**

- $\mathcal{L}_g(\text{ask, tell}) \nsubseteq \mathcal{L}_r(\text{ask, tell})$
- Inability for $\mathcal{L}_r(\text{ask, tell})$ to atomically test the presence of two distinct tokens $a$ and $b$.

- Assume $AB = [\text{ask}(a) \rightarrow \text{ask}(b)]$ and $C(AB)$ a coder (in $\mathcal{L}_r(\text{ask, tell})$)
- $C(AB)$ in general form:

\[
\begin{align*}
tell(t_1) & ; A_1 + \cdots + \text{tell}(t_p) ; A_p \\
+ \text{ask}(u_1) & ; B_1 + \cdots + \text{ask}(u_q) ; B_q \\
+ gp_1 & ; C_1 + \cdots + gp_r ; C_r
\end{align*}
\]
Expressiveness - Proof example

Propositions

- $\mathcal{L}_g(\text{ask, tell}) \not\subseteq \mathcal{L}_r(\text{ask, tell})$
- Inability for $\mathcal{L}_r(\text{ask, tell})$ to atomically test the presence of two distinct tokens $a$ and $b$.

- Assume $AB = [\text{ask}(a) \rightarrow \text{ask}(b)]$ and $C(AB)$ a coder (in $\mathcal{L}_r(\text{ask, tell})$)
- $C(AB)$ in general form:

  $$
tell(t_1) ; A_1 + \cdots + tell(t_p) ; A_p \\
+ \text{ask}(u_1) ; B_1 + \cdots + \text{ask}(u_q) ; B_q \\
+ gp_1 ; C_1 + \cdots + gp_r ; C_r
$$
Propositions

- $L_g(\text{ask, tell}) \not\subseteq L_r(\text{ask, tell})$
- Inability for $L_r(\text{ask, tell})$ to atomically test the presence of two distinct tokens $a$ and $b$.

- Assume $AB = [\text{ask}(a) \rightarrow \text{ask}(b)]$ and $C(AB)$ a coder (in $L_r(\text{ask, tell})$)
- $C(AB)$ in general form:

$$tell(t_1) ; A_1 + \cdots + tell(t_p) ; A_p$$
$$+ \text{ask}(u_1) ; B_1 + \cdots + \text{ask}(u_q) ; B_q$$
$$+ gp_1 ; C_1 + \cdots + gp_r ; C_r$$
Expressiveness – Proof example

\[ \langle C([\text{tell}(a)]) \mid \emptyset \rangle \]
Expressiveness – Proof example

\[ \langle C(\text{[tell}(a)]) \mid \emptyset \rangle \rightarrow \cdots \rightarrow \langle E \mid \{a_1, \ldots, a_m\} \rangle \]
Expressiveness – Proof example

\[ \langle C([tell(a)]) | \emptyset \rangle \rightarrow \cdots \rightarrow \langle E | \{a_1, \cdots, a_m\} \rangle \]

\[ \langle C([tell(b)]) | \emptyset \rangle \]
Expressiveness – Proof example

\[
\langle C([\text{tell}(a)]) \mid \emptyset \rangle \rightarrow \cdots \rightarrow \langle E \mid \{ a_1, \cdots, a_m \} \rangle
\]

\[
\langle C([\text{tell}(b)]) \mid \emptyset \rangle \rightarrow \cdots \rightarrow \langle E \mid \{ b_1, \cdots, b_m \} \rangle
\]
Expressiveness – Proof example

\[ \langle C([\text{tell}(a)]) \mid \emptyset \rangle \rightarrow \ldots \rightarrow \langle E \mid \{a_1, \ldots, a_m\} \rangle \]

\[ \langle C([\text{tell}(b)]) \mid \emptyset \rangle \rightarrow \ldots \rightarrow \langle E \mid \{b_1, \ldots, b_m\} \rangle \]

\[ \langle C([\text{tell}(b)]) \mid \tau \rangle \]
Expressiveness – Proof example

\[ \langle C(\text{tell}(a)) \mid \emptyset \rangle \longrightarrow \cdots \longrightarrow \langle E \mid \{a_1, \ldots, a_m\} \rangle \]

\[ \langle C(\text{tell}(b)) \mid \emptyset \rangle \longrightarrow \cdots \longrightarrow \langle E \mid \{b_1, \ldots, b_m\} \rangle \]

\[ \langle C(\text{tell}(b)) \mid \tau \rangle \longrightarrow \cdots \longrightarrow \langle E \mid \tau \cup \{b_1, \ldots, b_n\} \rangle \]
Expressiveness – Proof example

\[ \langle C([\text{tell}(a)]) \mid \emptyset \rangle \rightarrow \cdots \rightarrow \langle E \mid \{a_1, \ldots, a_m\}\rangle \]

\[ \langle C([\text{tell}(b)]) \mid \emptyset \rangle \rightarrow \cdots \rightarrow \langle E \mid \{b_1, \ldots, b_m\}\rangle \]

\[ \langle C([\text{tell}(b)]) \mid \tau \rangle \rightarrow \cdots \rightarrow \langle E \mid \tau \cup \{b_1, \ldots, b_n\}\rangle \]

\[ \langle C([\text{tell}(a)]; [\text{tell}(b)]) \mid \emptyset \rangle \]
Expressiveness – Proof example

\[ \langle C([\text{tell}(a)]) \mid \emptyset \rangle \rightarrow \cdots \rightarrow \langle E \mid \{a_1, \ldots, a_m\} \rangle \]

\[ \langle C([\text{tell}(b)]) \mid \emptyset \rangle \rightarrow \cdots \rightarrow \langle E \mid \{b_1, \ldots, b_m\} \rangle \]

\[ \langle C([\text{tell}(b)]) \mid \tau \rangle \rightarrow \cdots \rightarrow \langle E \mid \tau \cup \{b_1, \ldots, b_n\} \rangle \]

\[ \langle C([\text{tell}(a)]; [\text{tell}(b)]) \mid \emptyset \rangle \rightarrow \cdots \rightarrow \langle C([\text{tell}(b)]) \mid \{a_1, \ldots, a_m\} \rangle \]
Expressiveness – Proof example

\[ \langle C([\text{tell}(a)]) \mid \emptyset \rangle \longrightarrow \cdots \longrightarrow \langle E \mid \{a_1, \cdots, a_m\} \rangle \]

\[ \langle C([\text{tell}(b)]) \mid \emptyset \rangle \longrightarrow \cdots \longrightarrow \langle E \mid \{b_1, \cdots, b_m\} \rangle \]

\[ \langle C([\text{tell}(b)]) \mid \tau \rangle \longrightarrow \cdots \longrightarrow \langle E \mid \tau \cup \{b_1, \cdots, b_n\} \rangle \]

\[ \langle C([\text{tell}(a)]; [\text{tell}(b)]) \mid \emptyset \rangle \longrightarrow \cdots \longrightarrow \langle C([\text{tell}(b)]) \mid \{a_1, \cdots, a_m\} \rangle \]

\[ \langle E \mid \{a_1, \cdots, a_m, b_1, \cdots, b_n\} \rangle \]
Expressiveness - Proof example

\[ u_i \text{’s } \notin \{a_1, \cdots, a_m\} \cup \{b_1, \cdots, b_n\} \]
Expressiveness - Proof example

\[ u_i \text{'s} \notin \{a_1, \ldots, a_m\} \cup \{b_1, \ldots, b_n\} \]

\[ \langle([\text{tell}(a)] ; [\text{tell}(b)] ; AB) \mid \emptyset \rangle \]
Expressiveness - Proof example

\[ u_i's \notin \{a_1, \ldots, a_m\} \cup \{b_1, \ldots, b_n\} \]

\[ \langle([tell(a)] ; [tell(b)] ; AB) \mid \emptyset\rangle \quad \cdots \quad \rightarrow \quad \langle E \mid \{a, b\}\rangle \]
Expressiveness - Proof example

\[ u_i \not\in \{a_1, \ldots, a_m\} \cup \{b_1, \ldots, b_n\} \]

\[ \langle ([\text{tell}(a)] ; [\text{tell}(b)] ; AB) | \emptyset \rangle \rightarrow \cdots \rightarrow \langle E | \{a, b\} \rangle \]

\[ C \]

\[ \langle C([\text{tell}(a)] ; [\text{tell}(b)] ; AB) | \emptyset \rangle \]
Expressiveness - Proof example

\[ u_i's \notin \{a_1, \cdots, a_m\} \cup \{b_1, \cdots, b_n\} \]

\[
\langle([\text{tell}(a)] ; [\text{tell}(b)] ; AB) \mid \emptyset\rangle \xrightarrow{\cdots} \langle E \mid \{a, b\}\rangle
\]

\[
\left\langle C([\text{tell}(a)] ; [\text{tell}(b)] ; AB) \mid \emptyset\right\rangle
\]

\[
\xrightarrow{C} \langle AB \mid \{a_1, \cdots, a_m, b_1, \cdots, b_n\}\rangle \not\rightarrow
\]

M. Barkallah – J.-M. Jacquet
Guarded Lists in Bach
**Without GL:** The graph shows the time values without GL for different cases.
With GL: The graph displays the time values with GL for different cases.
Comparison: The graph compares the time values with and without GL for different cases.
Conclusion

- Introduce a new construct called guarded list
- Establish an increase in expressiveness
- Propose a theory of refinement
- Show an increase in performance
Conclusion

- Introduce a new construct called guarded list
- Establish an increase in expressiveness
- Propose a theory of refinement
- Show an increase in performance
Conclusion

- Introduce a new construct called guarded list
- Establish an increase in expressiveness
- Propose a theory of refinement
- Show an increase in performance
Conclusion

- Introduce a new construct called guarded list
- Establish an increase in expressiveness
- Propose a theory of refinement
- Show an increase in performance