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

NADI Namur Digital Institute

nadi.unamur.be

UNIVERSITÉ DE NAMUR

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?

UNIVERSITÉ DE NAMUR

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?

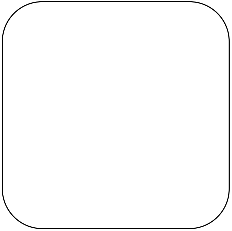
Coordination as a powerful paradigm . . . but in practice

- 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?

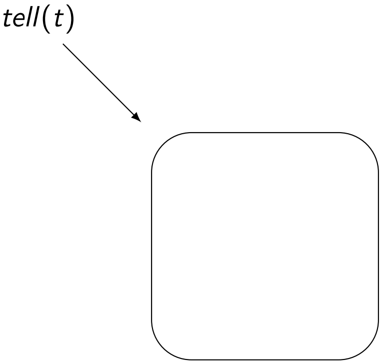
UNIVERSITÉ DE NAMUR

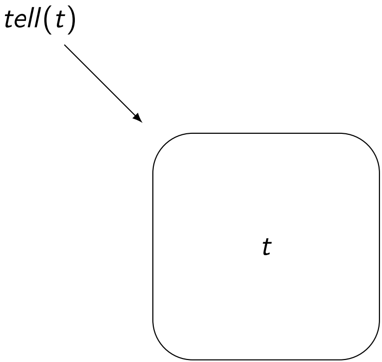
Coordination as a powerful paradigm . . . but in practice

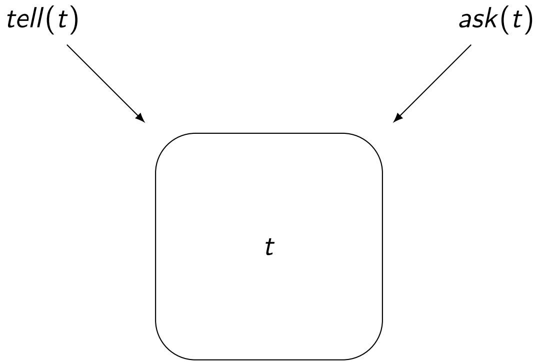
- a concurrent framework based on shared information
- clear separation between interactional and computational aspects
  - ☞ many models and languages
  - ☞ many theoretical pieces of work
  - ☞ many implementations
- in practice, how to construct programs?
  - ☞ how to describe (real-life) problems?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



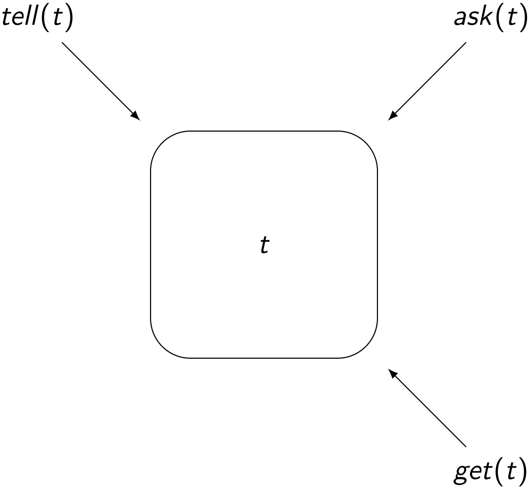
The Bach Coordination Language

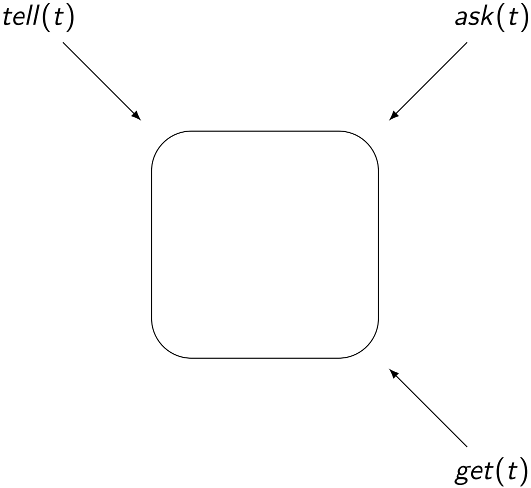



tell(t)
t

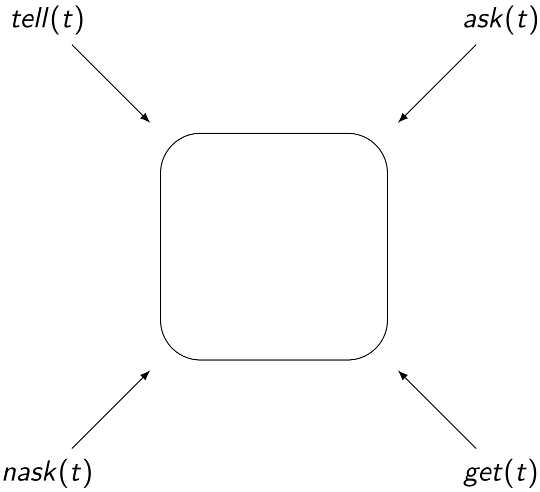

tell(t)
ask(t)
t

The Bach Coordination Language




tell(t)
ask(t)
get(t)

The Bach Coordination Language



Transition system

(T) $$\langle\, \mathrm{tell}(t) \mid \sigma \,\rangle \longrightarrow \langle\, E \mid \sigma \cup \{t\} \,\rangle$$

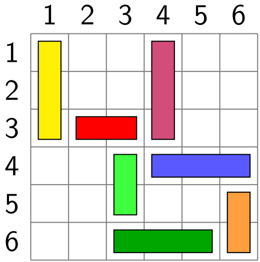
(A) $$\langle\, \mathrm{ask}(t) \mid \sigma \cup \{t\} \,\rangle \longrightarrow \langle\, E \mid \sigma \cup \{t\} \,\rangle$$

(G) $$\langle\, \mathrm{get}(t) \mid \sigma \cup \{t\} \,\rangle \longrightarrow \langle\, E \mid \sigma \,\rangle$$

(N) $$\frac{t \notin \sigma}{\langle\, \mathrm{nask}(t) \mid \sigma \,\rangle \longrightarrow \langle\, E \mid \sigma \,\rangle}$$

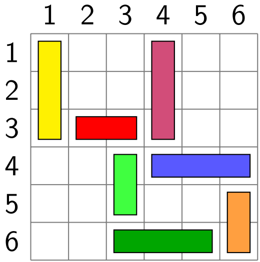
Rush hour as a running example

Modeling rush-hour



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

Modeling rush-hour



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

- finite sets

```
eset RCInt = { 1, 2, 3, 4, 5, 6}.
```

- maps and equations as rewriting rules

```
map down_truck : RCInt -> RCInt.
eqn down_truck(1) = 4. down_truck(2) = 5.
    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 `free(i,j)`

- finite sets

```
eset RCInt = { 1, 2, 3, 4, 5, 6}.
```

- maps and equations as rewriting rules

```
map down_truck : RCInt -> RCInt.
eqn down_truck(1) = 4. down_truck(2) = 5.
    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 [illegible]

Data

- finite sets

```
eset RCInt = { 1, 2, 3, 4, 5, 6}.
```

- maps and equations as rewriting rules

```
map down_truck : RCInt -> RCInt.
eqn down_truck(1) = 4. down_truck(2) = 5.
    down_truck(3) = 6.
```

- structured pieces of information
  - flat tokens: a, b, . . . , t, u, . . .
  - composed terms: $f(a_1, \ldots, a_n)$

- finite sets

```
eset RCInt = { 1, 2, 3, 4, 5, 6}.
```

- maps and equations as rewriting rules

```
map down_truck : RCInt -> RCInt.
eqn down_truck(1) = 4. down_truck(2) = 5.
    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 `free(a,g)`

Data

- finite sets

```
eset RCInt = { 1, 2, 3, 4, 5, 6}.
```

- maps and equations as rewriting rules

```
map down_truck : RCInt -> RCInt.
eqn down_truck(1) = 4. down_truck(2) = 5.
    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 free(i,j)

Data

- finite sets

```
eset RCInt = { 1, 2, 3, 4, 5, 6}.
```

- maps and equations as rewriting rules

```
map down_truck : RCInt -> RCInt.
eqn down_truck(1) = 4. down_truck(2) = 5.
    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 `free(i,j)`

Data

- finite sets

```
eset RCInt = { 1, 2, 3, 4, 5, 6}.
```

- maps and equations as rewriting rules

```
map down_truck : RCInt -> RCInt.
eqn down_truck(1) = 4. down_truck(2) = 5.
    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 `free(i,j)`

Data

- finite sets

```
eset RCInt = { 1, 2, 3, 4, 5, 6}.
```

- maps and equations as rewriting rules

```
map down_truck : RCInt -> RCInt.
eqn down_truck(1) = 4. down_truck(2) = 5.
    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 `free(i,j)`

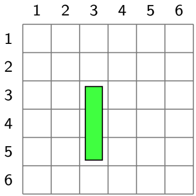
Agents

$$
\begin{aligned}
A \quad ::= \quad & Prim \mid Proc \mid \\
& A \,;\, A \mid A \parallel A \mid A + A \mid \\
& C \rightarrow A \diamond A \mid \sum_{e \in S} A_e
\end{aligned}
$$

Agents

$$
\begin{aligned}
A \quad ::= \quad & Prim \mid Proc \mid \\
& A \,;\, A \mid A \,||\, A \mid A \,+\, A \mid \\
& C \rightarrow A \diamond A \mid \sum_{e \in S} A_e
\end{aligned}
$$

where *Prim* represents a primitive, *Proc* a procedure call, C a condition, e a variable and S a set.




```
eset RCInt = { 1, 2, 3, 4, 5, 6}.
     Colors = { yellow, green, blue, purple, red, orange}.

proc VerticalTruck(r: RCInt, c: RCInt, p: Colors) =
       ( (r>1 & r<5) -> ( get(free(pred(r),c));
                          moveTruck(pred(r),c,p);
                          tell(free(succ(succ(r)),c));
                          VerticalTruck(pred(r),c,p) ))
       +
       ( (r<4) -> ( get(free(down_truck(r),c));
                    moveTruck(succ(r),c,p);
                    tell(free(r,c));
                    VerticalTruck(succ(r),c,p) )).
```

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:

$$TF \ ::= \ PF \mid Next\ TF \mid PF\ Until\ TF$$

- Reach formulae:

$$Reach\,(\#out = 1) \ \equiv \ true\ Until\,(\#out = 1)$$

Model checking

- Key information on the store: $\#\mathit{free}(1,1)$
- Basic formulae: equalities or inequalities involving integers and key information
  - ☞ $\#\mathit{free}(1,1) = 3$
- Propositional state formulae: combination of basic formulae by usual Boolean operators
- Linear temporal logic fragment:

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

- Reach formulae:

  $$\mathit{Reach}\,(\#\mathit{out} = 1) \ \equiv \ \mathit{true}\ \mathit{Until}\,(\#\mathit{out} = 1)$$

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:

$$TF ::= PF \mid Next\ TF \mid PF\ Until\ TF$$

- Reach formulae:

$$Reach\,(\#out = 1) \equiv true\ Until\,(\#out = 1)$$

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:

  $$TF \ ::= \ PF \mid Next\ TF \mid PF\ Until\ TF$$

- Reach formulae:

  $$Reach\,(\#out = 1) \ \equiv \ true\ Until\,(\#out = 1)$$

Model checking

- Key information on the store: $\#\mathit{free}(1,1)$
- Basic formulae: equalities or inequalities involving integers and key information
  - ☞ $\#\mathit{free}(1,1) = 3$
- Propositional state formulae: combination of basic formulae by usual Boolean operators
- Linear temporal logic fragment:

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

- Reach formulae:

$$\mathit{Reach}\,(\#\mathit{out} = 1) \ \equiv \ \mathit{true}\ \mathit{Until}\,(\#\mathit{out} = 1)$$

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:

$$TF \; ::= \; PF \mid Next\ TF \mid PF\ Until\ TF$$

- Reach formulae:

$$Reach\,(\#out = 1) \;\equiv\; true\ Until\ (\#out = 1)$$

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:

  $$TF \ ::= \ PF \mid Next\ TF \mid PF\ Until\ TF$$

- Reach formulae:

  $$Reach\,(\#out = 1) \ \equiv \ true\ Until\,(\#out = 1)$$

UNIVERSITÉ DE NAMUR

Model checking

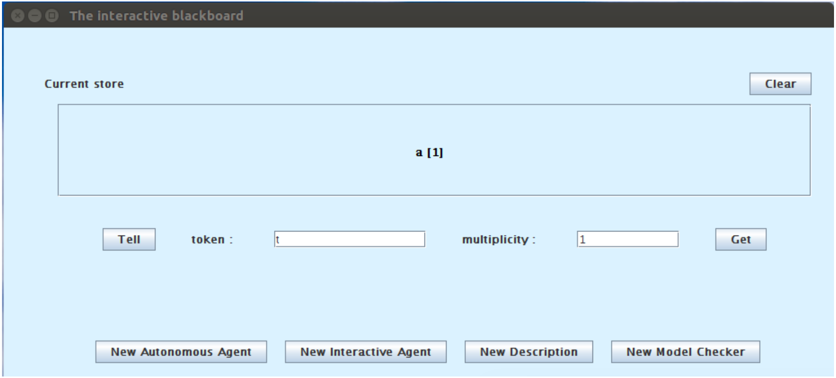
- Key information on the store: $\#\mathit{free}(1,1)$
- Basic formulae: equalities or inequalities involving integers and key information
  - ☞ $\#\mathit{free}(1,1) = 3$
- Propositional state formulae: combination of basic formulae by usual Boolean operators
- Linear temporal logic fragment:

$$TF \;::=\; PF \mid \mathit{Next}\ TF \mid PF\ \mathit{Until}\ TF$$

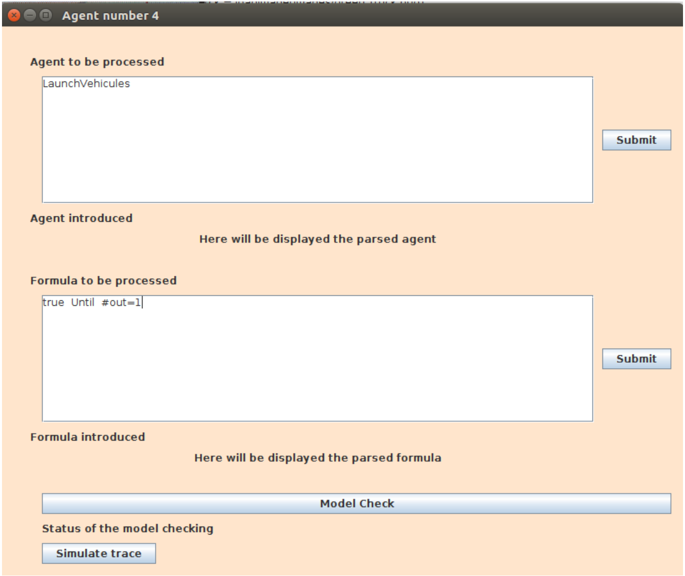
- Reach formulae:

$$\mathit{Reach}\,(\#\mathit{out} = 1) \;\equiv\; \mathit{true}\ \mathit{Until}\,(\#\mathit{out} = 1)$$

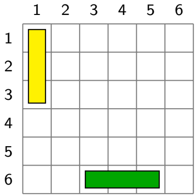
Scan & Anemone



Scan & Anemone



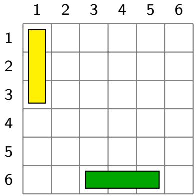
Problem




```
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))
```

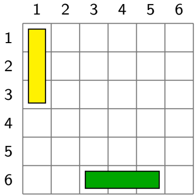
Problem




```
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))
```

Problem




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

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

$[p \rightarrow p_1, \cdots, p_n]$ where p, p_1, ..., p_n are primitives

$$\textbf{(Le)} \quad \langle\, [] \mid \sigma \,\rangle \longrightarrow \langle\, E \mid \sigma \,\rangle$$

$$\textbf{(Ln)} \quad \frac{\langle\, p \mid \sigma \,\rangle \longrightarrow \langle\, E \mid \tau \,\rangle,\ \langle\, L \mid \tau \,\rangle \longrightarrow^* \langle\, E \mid \phi \,\rangle}{\langle\, [p|L] \mid \sigma \,\rangle \longrightarrow \langle\, E \mid \phi \,\rangle}$$

$$\textbf{(GL)} \quad \frac{\langle\, p \mid \sigma \,\rangle \longrightarrow \langle\, E \mid \tau \,\rangle,\ \langle\, L \mid \tau \,\rangle \longrightarrow^* \langle\, E \mid \phi \,\rangle}{\langle\, [p \rightarrow L] \mid \sigma \,\rangle \longrightarrow \langle\, E \mid \phi \,\rangle}$$

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

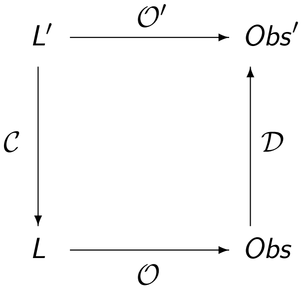
Objectives

- 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'

Expressiveness - Proof example

Propositions

- $\mathcal{L}_g(ask, tell) \not\leq \mathcal{L}_r(ask, tell)$
- 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 $\mathcal{C}(AB)$ a coder (in $\mathcal{L}_r(ask, tell)$)
- $\mathcal{C}(AB)$ in general form:

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

Propositions

- $\mathcal{L}_g(ask, tell) \not\leq \mathcal{L}_r(ask, tell)$
- 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 $\mathcal{C}(AB)$ a coder (in $\mathcal{L}_r(ask, tell)$)
- $\mathcal{C}(AB)$ in general form:

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

Expressiveness - Proof example

Propositions

- $\mathcal{L}_g(ask, tell) \not\leq \mathcal{L}_r(ask, tell)$
- 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 $\mathcal{C}(AB)$ a coder (in $\mathcal{L}_r(ask, tell)$)
- $\mathcal{C}(AB)$ in general form:

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

Propositions

- $\mathcal{L}_g(ask, tell) \not\leq \mathcal{L}_r(ask, tell)$
- 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 $\mathcal{C}(AB)$ a coder (in $\mathcal{L}_r(ask, tell)$)
- $\mathcal{C}(AB)$ in general form:

$$\begin{aligned} & tell(t_1) \;;\; A_1 + \cdots + tell(t_p) \;;\; A_p \\ & \quad + ask(u_1) \;;\; B_1 + \cdots + ask(u_q) \;;\; B_q \\ & \quad + gp_1 \;;\; C_1 + \cdots + gp_r \;;\; C_r \end{aligned}$$

Propositions

- $\mathcal{L}_g(ask, tell) \not\leq \mathcal{L}_r(ask, tell)$
- 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 $\mathcal{C}(AB)$ a coder (in $\mathcal{L}_r(ask, tell)$)
- $\mathcal{C}(AB)$ in general form:

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

Expressiveness – Proof example

$\langle \mathcal{C}([tell(a)]) \mid \emptyset \rangle$

Expressiveness – Proof example

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

Expressiveness – Proof example

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

$$\langle \mathcal{C}([tell(b)]) \mid \emptyset \rangle$$

Expressiveness – Proof example

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

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

Expressiveness – Proof example

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

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

$$\langle \mathcal{C}([tell(b)]) \mid \tau \rangle$$

Expressiveness – Proof example

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

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

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

Expressiveness – Proof example

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

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

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

$$\langle \mathcal{C}([tell(a)]; [tell(b)]) \mid \emptyset \rangle$$

Expressiveness – Proof example

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

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

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

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

Expressiveness – Proof example

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

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

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

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

$$\downarrow$$

$$\langle E \mid \{a_1, \cdots, a_m, b_1, \cdots, b_n\} \rangle$$

Expressiveness - Proof example

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

Expressiveness - Proof example

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

$$\langle ([tell(a)] \; ; \; [tell(b)] \; ; \; AB) \mid \emptyset \rangle$$

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

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

Expressiveness - Proof example

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

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

$$\Big\downarrow \mathcal{C}$$

$$\langle\mathcal{C}([tell(a)]\ ;\ [tell(b)]\ ;\ AB) \mid \emptyset\rangle$$

Expressiveness - Proof example

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

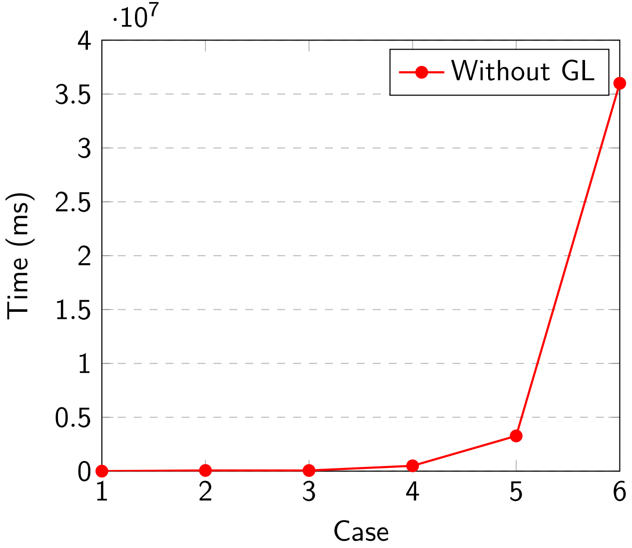
$$\langle([tell(a)] \;;\; [tell(b)] \;;\; AB) \mid \emptyset\rangle \longrightarrow \cdots \longrightarrow \langle E \mid \{a, b\}\rangle$$

$$\Big\downarrow \mathcal{C}$$

$$\langle\mathcal{C}([tell(a)] \;;\; [tell(b)] \;;\; AB) \mid \emptyset\rangle$$

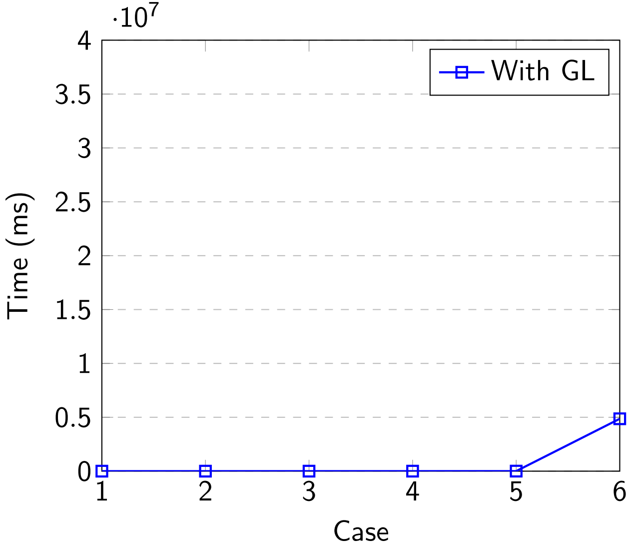
$$\longrightarrow \langle AB \mid \{a_1, \cdots, a_m, b_1, \cdots, b_n\}\rangle \not\longrightarrow$$

Performance



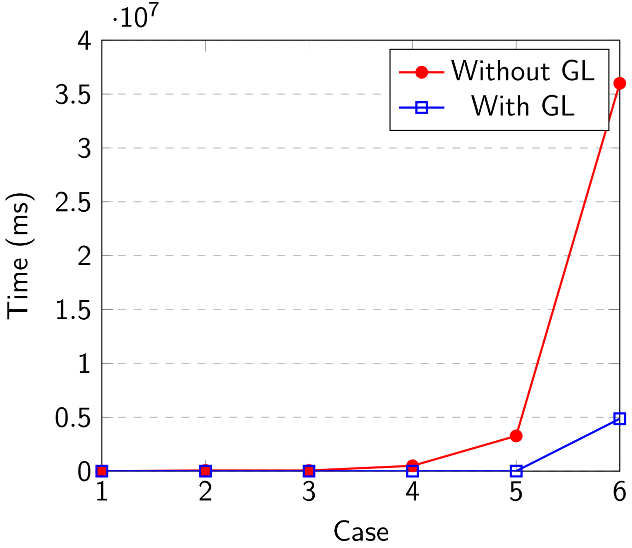
Without GL: The graph shows the time values without GL for different cases.

Performance



With GL: The graph displays the time values with GL for different cases.

Performance



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


THANK
YOU