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

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

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?

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?

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?

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?

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?

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?

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?

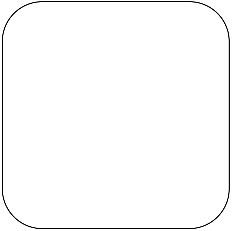
a concurrent framework based on shared information

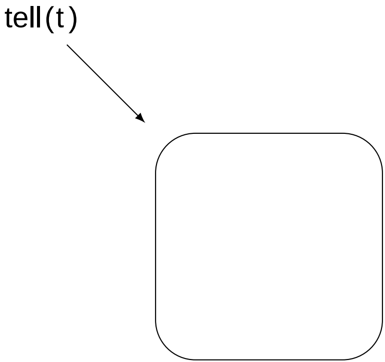
clear separation between interactional and computational aspects

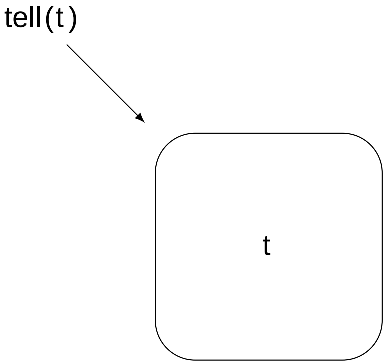
+ many models and languages
+ many theoretical pieces of work
+ many implementations

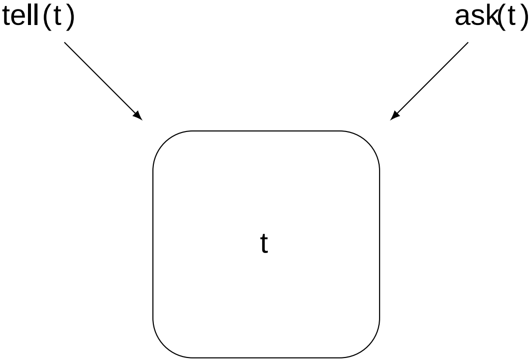
in practice, how to construct programs?

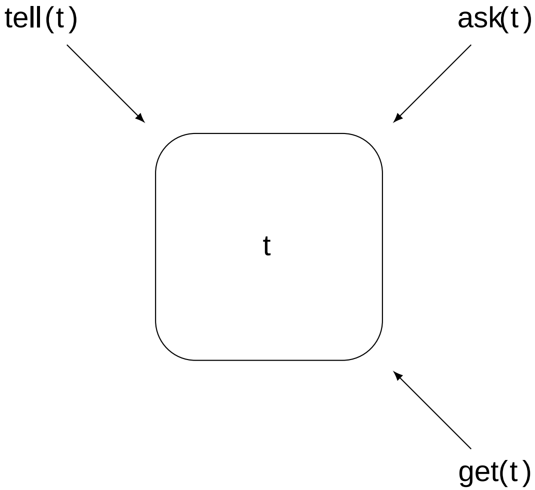
+ how to describe (real-life) problems?
+ how to reason on the programs?
+ how to be sure that what is described by the programs corresponds to what has to be modelled?

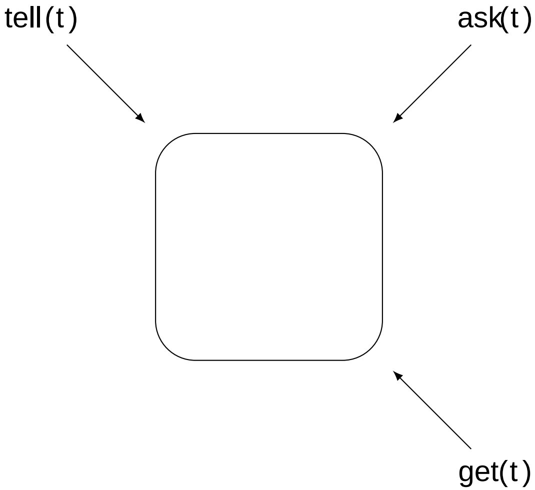


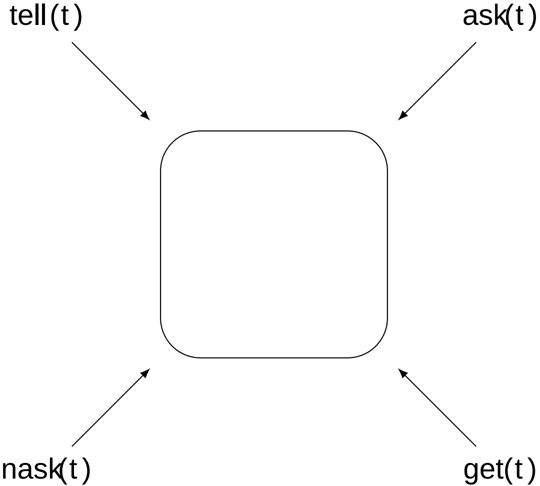

tell(t)


tell(t)
t


tell(t)
ask(t)
t


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


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

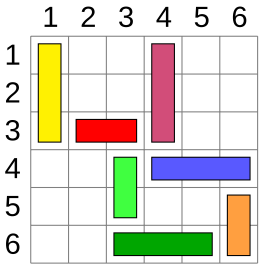

tell(t)
ask(t)
nask(t)
get(t)

(T) htell(t) j i ! h E j [f tg i

(A) hask(t) j [f tg i ! h E j [f tg i

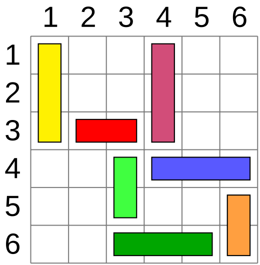
(G) hget(t) j [f tg i ! h E j i

(N) t 62 / hnask(t) j i ! h E j i



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

nite sets

```
eset RCInt = f 1, 2, 3, 4, 5, 6g.
```

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

- at tokens: a, b, . . . , t , u, . . .
- composed terms $f(a_1; \ldots; a_n)$
  - free places asfree(t; j)

nite sets

```
eset RCInt = f 1, 2, 3, 4, 5, 6g.
```

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

at tokens: a, b, . . . , t , u, . . .

composed terms $f(a_1; : : : ; a_n)$

free places asfree(t;)

nite sets

```
eset RCInt = f 1, 2, 3, 4, 5, 6g.
```

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

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

nite sets

```
eset RCInt = f 1, 2, 3, 4, 5, 6g.
```

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

- at tokens: a, b, . . . , t, u, . . .
- composed terms $f(a_1, \ldots, a_n)$
  - free places as [illegible]

nite sets

```
eset RCInt = f 1, 2, 3, 4, 5, 6g.
```

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

at tokens: a, b, ..., t, u, ...
composed terms $f(a_1; \ldots; a_n)$

+ free places as free(i,j)

nite sets

```
eset RCInt = f 1, 2, 3, 4, 5, 6g.
```

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

at tokens: a, b, . . . , t , u, . . .

composed terms $f(a_1; : : : ; a_n)$

+ free places as free(i,j)

nite 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

- at tokens: a, b, . . . , t , u, . . .
- composed terms $f(a_1, \ldots, a_n)$
  - + free places as free(i,j)

nite sets

```
eset RCInt = f 1, 2, 3, 4, 5, 6g.
```

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

at tokens: a, b, . . . , t , u, . . .

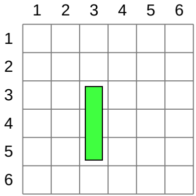
composed terms $f(a_1; : : : ; a_n)$

+ free places as free(i,j)

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

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

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




```
eset RCInt = f 1, 2, 3, 4, 5, 6g.
     Colors = f yellow, green, blue, purple, red, orangeg.

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) )).
```

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:

$$Reach(\#out = 1) \qquad true\ Until\ (\#out = 1)$$

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:

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

Reach formulae:

$$Reach(\#out = 1) \quad true\ Until\ (\#out = 1)$$

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 j Next TF j PF Until TF

Reach formulae:

Reach(# out = 1) 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 j Next TF j PF Until TF

Reach formulae:

Reach(# out = 1) 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:

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

Reach formulae:

$$Reach(\#\ out = 1) \quad true\ Until\ (\#\ out = 1)$$

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 j Next TF j PF Until TF

Reach formulae:

Reach(# out = 1) 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 j Next TF j PF Until TF

Reach formulae:

Reach(# out = 1)   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

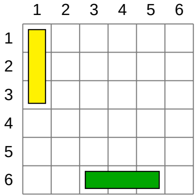
Propositional state formulae: combination of basic formulae by usual Boolean operators

Linear temporal logic fragment:

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

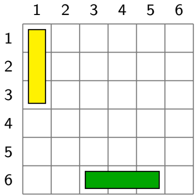
Reach formulae:

Reach(# out = 1) true Until (# out = 1)



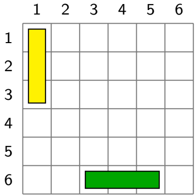

```
get(free(pred(r),c));
move(truck_img(c),pred(r),c);
tell(free(succ(succ(r)),c))
```

```
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))
```

```
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)) ]
```

The construct

$[p\ \ !\ \ p_1;\qquad ;p_n]$ where p, p_1, . . . , p_n are primitives

(Le) $$h\,[]\,j\quad i\quad !\ h\,E\,j\quad i$$

(Ln) $$\frac{h\,p\,j\quad i\quad !\ h\,E\,j\quad i;\ h\,L\,j\quad i\quad !\quad h\,E\,j\quad i}{h\,[pjL]\,j\quad i\quad !\ h\,E\,j\quad i}$$

(GL) $$\frac{h\,p\,j\quad i\quad !\ h\,E\,j\quad i;\ h\,L\,j\quad i\quad !\quad h\,E\,j\quad i}{h\,[p\ \ !\ \ L]\,j\quad i\quad !\ h\,E\,j\quad i}$$

Introduce a new construct called guarded list

Establish an increase of expressiveness

Propose a theory of refinement

Show an increase of performance

Introduce a new construct called guarded list

Establish an increase of expressiveness

Propose a theory of refinement

Show an increase of performance

Introduce a new construct called guarded list

Establish an increase of expressiveness

Propose a theory of refinement

Show an increase of performance

Introduce a new construct called guarded list

Establish an increase of expressiveness

Propose a theory of refinement

Show an increase of performance

$$
\begin{tikzcd}
L^0 \arrow[r, "O^0"] \arrow[d, "C"'] & Obs^0 \\
L \arrow[r, "O"'] & Obs \arrow[u, "D"']
\end{tikzcd}
$$

L embeds L^0

Propositions

$L_g(ask; tell)$ 6 $L_r(ask; tell)$

Inability for $L_r(ask; tell)$ to atomically test the presence of two distinct tokens a and b.

Assume $AB = [ask(a) \; ! \; ask(b)]$ and $C(AB)$ a coder (in $L_r(ask; tell)$)

$C(AB)$ in general form:

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

Propositions

$L_g(ask;tell)\ 6\ \ L_r(ask;tell)$

Inability for $L_r(ask;tell)$ to atomically test the presence of two distinct tokens a and b.

Assume $AB = [ask(a)\ \ !\ \ ask(b)]$ and $C(AB)$ a coder (in $L_r(ask;tell)$)

$C(AB)$ in general form:

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

[redacted]

[redacted]

- $L_g(ask; tell) \not\leq L_r(ask; tell)$
- Inability for $L_r(ask; tell)$ 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 $L_r(ask; tell)$)

$C(AB)$ in general form:

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

[redacted]

[redacted]

$L_g(ask, tell) \not\leq L_r(ask, tell)$

Inability for $L_r(ask, tell)$ 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 $L_r(ask, tell)$)

$C(AB)$ in general form:

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

[redacted]

$\mathcal{L}_g(ask, tell) \not\leq \mathcal{L}_r(ask, tell)$

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

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

$\mathcal{C}(AB)$ in general form:

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

hC([tell(a)]) j ;i

hC([tell(a)]) j ;i $\longrightarrow$! h E j f a$_1$; ; a$_m$gi

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

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

$$hC([tell(a)])\ j\ ;i \longrightarrow\ !\ h\ \ E\ j\ f\ a_1; \quad ; a_m gi$$

$$hC([tell(b)])\ j\ ;i \longrightarrow\ !\ h\ \ E\ j\ f\ b_1; \quad ; b_m gi$$

$$hC([tell(a)])\ j\ ;i \longrightarrow\ !\ h\ \ E\ j\ f\ a_1;\ \ \ ;a_m gi$$

$$hC([tell(b)])\ j\ ;i \longrightarrow\ !\ h\ \ E\ j\ f\ b_1;\ \ \ ;b_m gi$$

$$hC([tell(b)])\ j\ \ i$$

hC([tell(a)]) j ;i $\longrightarrow$! h E j f a_1; ; a_m gi

hC([tell(b)]) j ;i $\longrightarrow$! h E j f b_1; ; b_m gi

hC([tell(b)]) j i $\longrightarrow$! h E j [f b_1; ; b_n gi hC([tell(a)]) j ;i $\longrightarrow$! h E j f a_1; ; $a_m$gi

hC([tell(b)]) j ;i $\longrightarrow$! h E j f b_1; ; $b_m$gi

hC([tell(b)]) j i $\longrightarrow$! h E j [f b_1; ; $b_n$gi

hC([tell(a)]; [tell(b)]) j ;i

$$hC([tell(a)]) \; j \; ;i \longrightarrow \; ! \; h \quad E \; j \; f \; a_1; \quad ; a_m gi$$

$$hC([tell(b)]) \; j \; ;i \longrightarrow \; ! \; h \quad E \; j \; f \; b_1; \quad ; b_m gi$$

$$hC([tell(b)]) \; j \quad i \longrightarrow \; ! \; h \quad E \; j \quad [\; f \quad b_1; \quad ; b_n gi$$

$$hC([tell(a)]; [tell(b)]) \; j \; ;i \longrightarrow \; ! \; hC \quad ([tell(b)]) \; j \; f \; a_1; \quad ; a_m gi$$

hC([tell(a)]) j ;i ⟶ ! h E j f a_1; ; $a_m$gi

hC([tell(b)]) j ;i ⟶ ! h E j f b_1; ; $b_m$gi

hC([tell(b)]) j i ⟶ ! h E j [f b_1; ; $b_n$gi

hC([tell(a)]; [tell(b)]) j ;i ⟶ ! hC ([tell(b)]) j f a_1; ; $a_m$gi

hE j f a_1; ; a_m; b_1; ; $b_n$gi

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

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

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

u_i 's 2= f a_1; ; a_m g [f b_1; ; b_n g

h([tell(a)] ; [tell(b)] ; AB) j ;i $\longrightarrow$! h E j f a; bgi

u_i 's 2= f a_1; ; a_m g [f b_1; ; b_n g

h([tell(a)] ; [tell(b)] ; AB) j ;i ⟶ ! h E j f a; bgi

↓ C

hC([tell(a)] ; [tell(b)] ; AB) j ;i

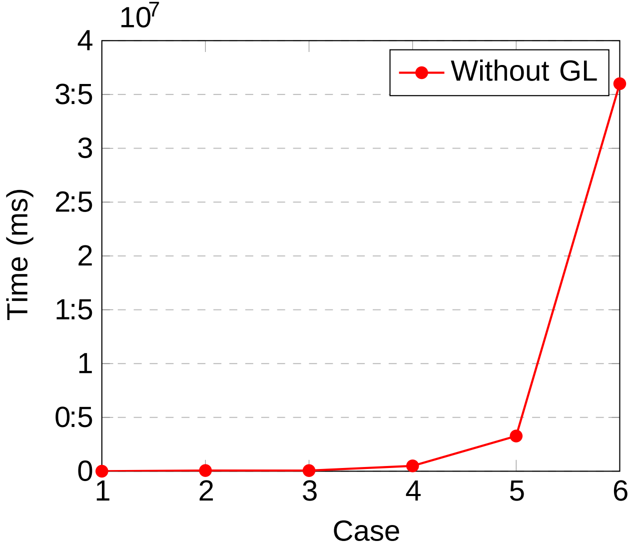
u_i's 2= f a_1; ; a_m g [f b_1; ; b_n g

h([tell(a)] ; [tell(b)] ; AB) j ;i $\longrightarrow$! h E j f a; bgi

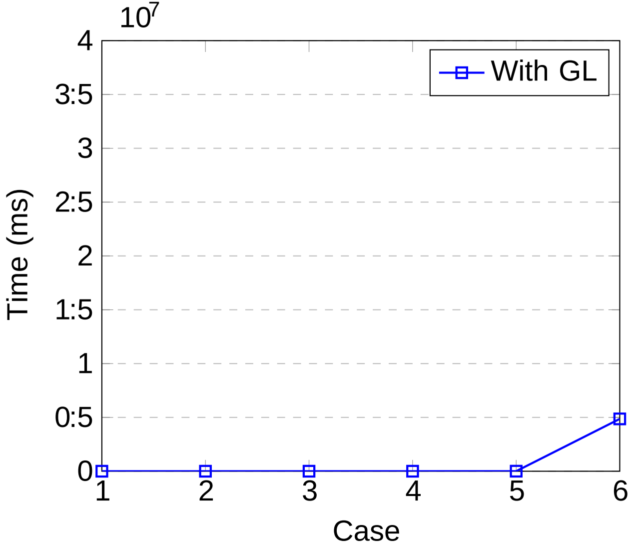
$\Big\downarrow C$

hC([tell(a)] ; [tell(b)] ; AB) j ;i

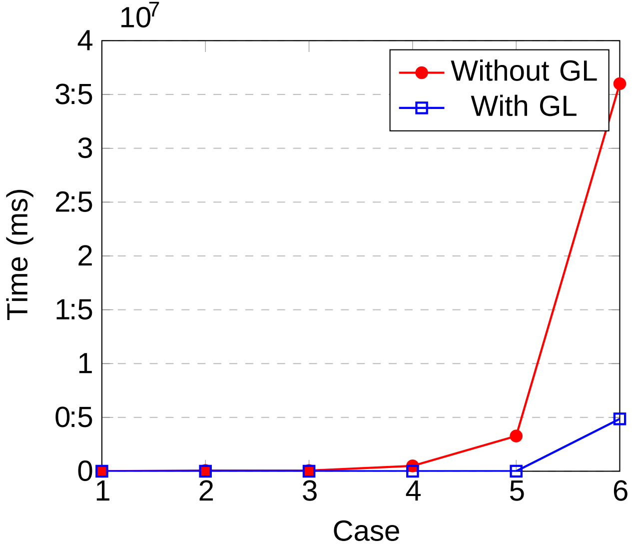
$\longrightarrow$ hAB j f a_1; ; a_m; b_1; ; b_n gi 6 !



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



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



Comparison: The graph compares the time values with and without GL for di erent cases.

Introduce a new construct called guarded list

Establish an increase in expressiveness

Propose a theory of re nement

Show an increase in performance

Introduce a new construct called guarded list

Establish an increase in expressiveness

Propose a theory of re nement

Show an increase in performance

Introduce a new construct called guarded list

Establish an increase in expressiveness

Propose a theory of re nement

Show an increase in performance

Introduce a new construct called guarded list

Establish an increase in expressiveness

Propose a theory of re nement

Show an increase in performance