A Predicate Transformer for Choreographies [ESOP'22]

Sung-Shik Jongmans[1,2] and Petra van den Bos[3]

[1] Open University of the Netherlands
[2] Centrum Wiskunde & Informatica (CWI)
[3] University of Twente

Long-term **research aim**:

Develop theoretical **foundations**
and practical **tools** to make
concurrent programming easier

Long-term **research aim:**

Develop theoretical **foundations** and practical **tools** to make **concurrent programming easier**

This paper:

1.

Communication via message passing

(channel-based)

Long-term **research aim:**

Develop theoretical **foundations**
and practical **tools** to make
concurrent programming easier

This paper:

1.
Communication via message passing
(channel-based)

2.
X-by-construction

Long-term **research aim:**

Develop theoretical **foundations** and practical **tools** to make **concurrent programming easier**

This paper:

1.

Communication via message passing

(channel-based)

2.

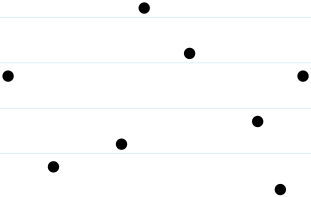
X-by-construction

3.

$X \in \{$ functional correctness (pre/post), deadlock freedom $\}$

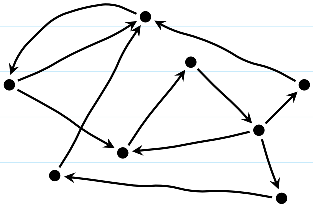
Suppose that a program consists of:

1. ???
2. ???
3. ???



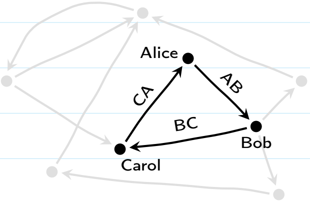
Suppose that a program consists of:

1. processes 2. ??? 3. ???



Suppose that a program consists of:

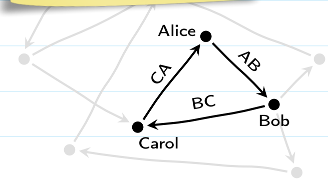
1. processes 2. channels 3. ???



Suppose that a program consists of:

1. processes 2. channels 3. “local programs”

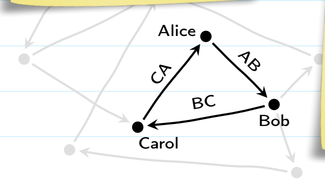
$$\underbrace{\mathsf{CA}?\mathtt{x}}_{\text{receive from Carol}}\ ;\ \overbrace{\mathsf{AB}!\,(\mathtt{x}*6)}^{\text{send to Bob}}$$



Suppose that a program consists of:

1. processes
2. channels
3. “local programs”

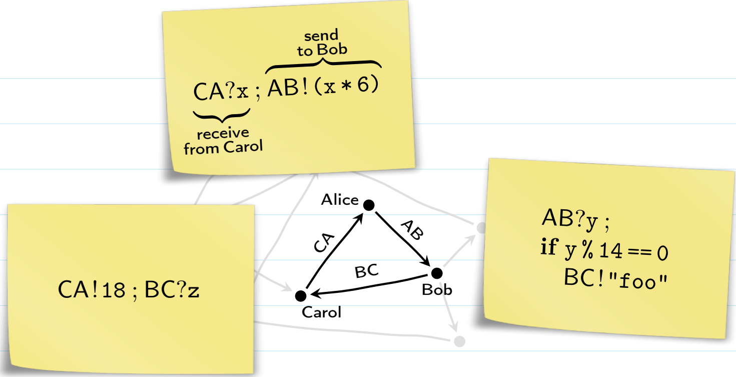
$$\underbrace{\mathsf{CA}?\mathtt{x}}_{\text{receive from Carol}}\ ;\ \overbrace{\mathsf{AB}!\,(\mathtt{x} * 6)}^{\text{send to Bob}}$$



```
AB?y ;
if y % 14 == 0
  BC!"foo"
```

Suppose that a program consists of:

1. processes 2. channels 3. “local programs”



$\underbrace{\mathsf{CA?}\mathtt{x}}_{\text{receive from Carol}}\ ;\ \overbrace{\mathsf{AB!}\,(\mathtt{x} * 6)}^{\text{send to Bob}}$

CA!18 ; BC?z

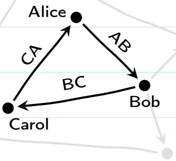
AB?y ;
if y % 14 == 0
  BC!"foo"

Suppose that a program consists of:

1. processes 2. channels 3. “local programs”

$$\underbrace{\mathsf{CA}?\mathtt{x}}_{\text{receive from Carol}}\ ;\ \overbrace{\mathsf{AB}!\,(\mathtt{x}*6)}^{\text{send to Bob}}$$

```
CA!18 ; BC?z
```



```
AB?y ;
if y % 14 == 0
   BC!"foo"
```

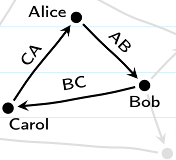
How to prove that the program is functionally correct and deadlock-free?

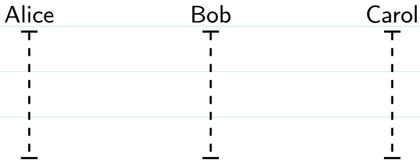
Not so easy... (even if we ignore functional correctness)

$\underbrace{\texttt{CA?x}}_{\text{receive from Carol}}\ ;\ \overbrace{\texttt{AB!(x*6)}}^{\text{send to Bob}}$

CA!18 ; BC?z

AB?y ;
if y % 14 == 0
  BC!"foo"

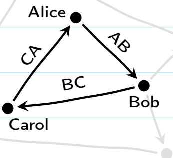




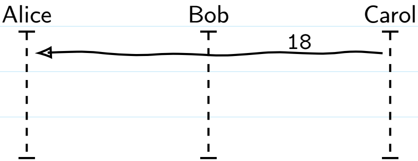
(intuitively: global analysis)

$\underbrace{\cancel{\mathsf{CA?x}}}_{\text{receive from Carol}}\ ;\ \overbrace{\mathsf{AB!(x*6)}}^{\text{send to Bob}}$

~~CA!18~~ ; BC?z



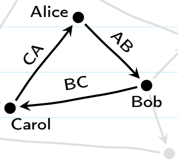
```
AB?y ;
if y % 14 == 0
  BC!"foo"
```



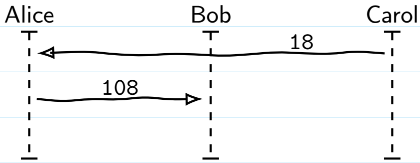
(intuitively: global analysis)

$$\underbrace{\cancel{CA?x}}_{\text{receive from Carol}}\ ;\ \overbrace{\cancel{AB!(x*6)}}^{\text{send to Bob}}$$

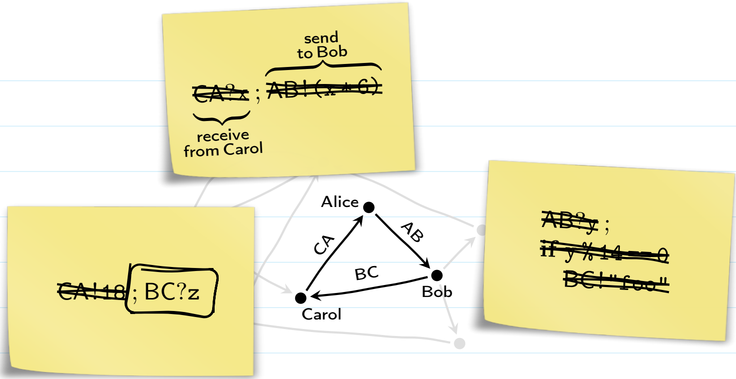
~~CA!18~~ ; BC?z



~~AB?y~~ ;
if y % 14 == 0
  BC!"foo"



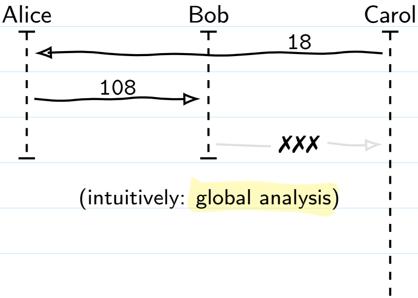
(intuitively: global analysis)



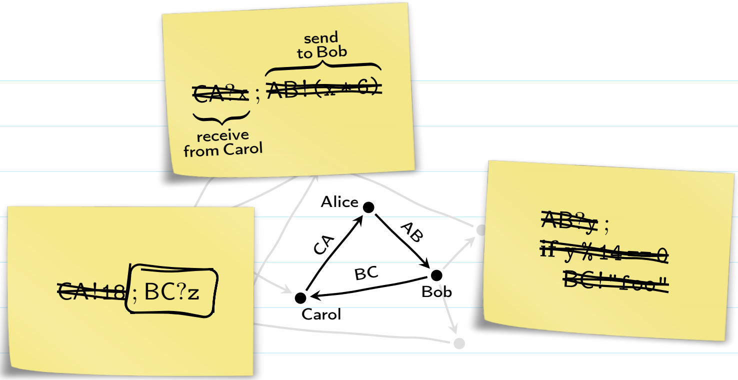
~~CA?x~~ ; ~~AB!(x * 6)~~

~~CA!18~~; $\boxed{\text{BC?z}}$

~~AB?y~~ ;
~~if y % 14 == 0~~
~~BC!"foo"~~



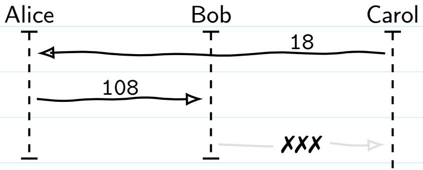
(intuitively: global analysis)



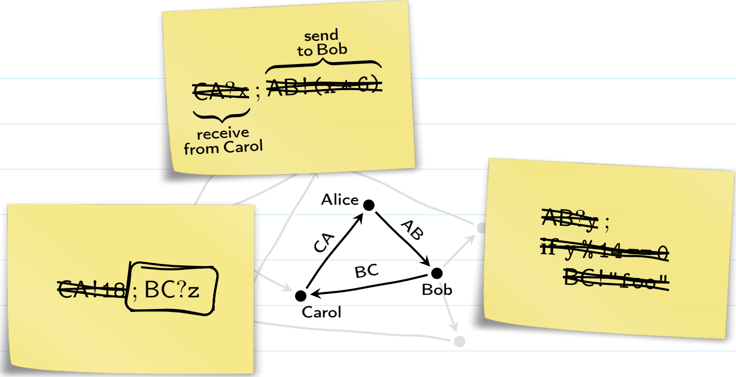
$\underbrace{\cancel{\mathsf{CA?x}}}_{\text{receive from Carol}}\ ;\ \overbrace{\cancel{\mathsf{AB!(x*6)}}}^{\text{send to Bob}}$

~~CA!18~~ $\boxed{\text{; BC?z}}$

~~AB?y~~ ;
~~if y % 14 == 0~~
~~BC!"foo"~~



(intuitively: $\underbrace{\text{global analysis}}_{\text{tedious, error-prone, …}}$)

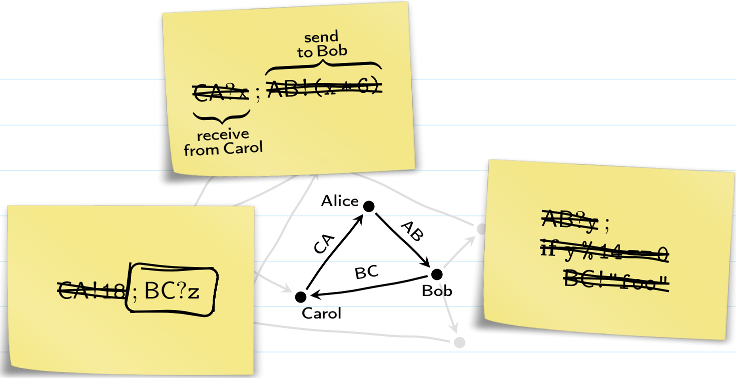


$$\underbrace{\cancel{\mathrm{CA?}x}}_{\text{receive from Carol}} ; \overbrace{\cancel{\mathrm{AB!}(x*6)}}^{\text{send to Bob}}$$

$\cancel{\mathrm{CA!}18}\boxed{; \mathrm{BC?}z}$

~~AB?y~~ ;
~~if y % 14 == 0~~
~~BC!"foo"~~

How to prove that the program is functionally correct and deadlock-free?



How to prove that the program is functionally correct and deadlock-free?

This paper: Choreographic programming

Choreographic programming in a nutshell:

1. ???

2. ???

[Carbone et al., ESOP'07/TOPLAS; Carbone & Montesi, POPL'13]

Choreographic programming in a nutshell:

1. Write global program G (manually)

2. ???

[Carbone et al., ESOP'07/TOPLAS; Carbone & Montesi, POPL'13]

Choreographic programming in a nutshell:

1. Write global program G (manually)

Theorem: G is deadlock-free

2. ???

[Carbone et al., ESOP'07/TOPLAS; Carbone & Montesi, POPL'13]

Choreographic programming in a

1. Write global program G (manua

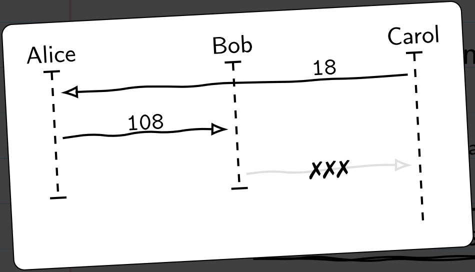
Theorem: G is de

2. ???

[Carbone et al., ESOP'07/TOPLAS; Carbone & Montesi, POPL'13]

1. Write global program

$$
\begin{aligned}
G = {} & \mathsf{C}.18 \rightarrowtriangle \mathsf{A}.\mathtt{x}\ ; \\
& \mathsf{A}.(\mathtt{x} * 6) \rightarrowtriangle \mathsf{B}.\mathtt{y}\ ; \\
& \textbf{if}\ \mathsf{B}.(\mathtt{y}\,\%\,14 == 0) \\
& \quad \mathsf{B}.\texttt{"foo"} \rightarrowtriangle \mathsf{C}.\mathtt{z}
\end{aligned}
$$



n a

anua

s de

1. Write global program

$$
\begin{aligned}
G = {} & \mathsf{C}.18 \rightarrow \mathsf{A}.\mathtt{x}\ ;\\
& \mathsf{A}.(\mathtt{x} * 6) \rightarrow \mathsf{B}.\mathtt{y}\ ;\\
& \textbf{if}\ \mathsf{B}.(\mathtt{y}\,\%\,14 == 0)\\
& \quad \mathsf{B}.\texttt{"foo"} \rightarrow \mathsf{C}.\mathtt{z}
\end{aligned}
$$

2. ???

[Carbone et al., ESOP'07/TOPLAS; Carbone & Montesi, POPL'13]

Choreographic programming in a nutshell:

1. Write global program G (manually)

Theorem: G is deadlock-free

2. Decompose into local programs $L_1, \ldots, L_n$ (automatically)

[Carbone et al., ESOP'07/TOPLAS; Carbone & Montesi, POPL'13]

Choreographic programming in a nutshell:

1. Write global program G (manually)

Theorem: G is deadlock-free

2. Decompose into local programs $L_1, \ldots, L_n$ (automatically)

Theorem:
$G \approx L_1 | \cdots | L_n$

[Carbone et al., ESOP'07/TOPLAS; Carbone & Montesi, POPL'13]

Choreographic programming in a nutshell:

1. Write global program G (manually)

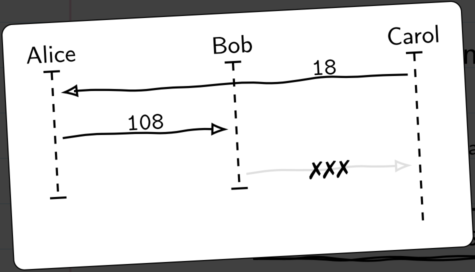
Theorem: G is deadlock-free

2. Decompose into local programs $L_1, \ldots, L_n$ (automatically)

Theorem:
$G \approx L_1 | \cdots | L_n$

Corollary:
$L_1 | \cdots | L_n$ is deadlock-free

[Carbone et al., ESOP'07/TOPLAS; Carbone & Montesi, POPL'13]



1. Write global program

```
G = C.18 → A.x ;
    A.(x * 6) → B.y ;
    if B.(y % 14 == 0)
       B."foo" → C.z
```

2. Decompose into local programs

$L_A = ???$

$L_B = ???$

$L_C = ???$

n a

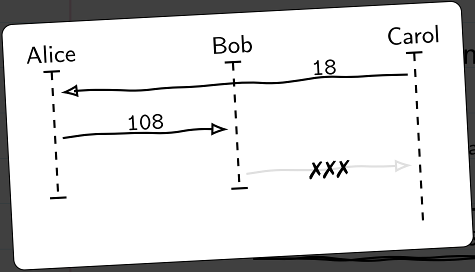
anua

s de

2. Dec … local programs $L_1, \ldots, L_n$ (automatically)

ee

, POPL'13]



n a

anua

s de

1. Write global program

$$
\begin{aligned}
G = {} & \mathsf{C}.18 \rightarrow \mathsf{A}.\mathtt{x}\ ; \\
& \mathsf{A}.(\mathtt{x} * 6) \rightarrow \mathsf{B}.\mathtt{y}\ ; \\
& \textbf{if}\ \mathsf{B}.(\mathtt{y}\,\%\,14 == 0) \\
& \quad \mathsf{B}.\texttt{"foo"} \rightarrow \mathsf{C}.\mathtt{z}
\end{aligned}
$$

2. Dec into local programs $L_1, \ldots, L_n$ (automatically)

2. Decompose into local programs

$$
\begin{aligned}
L_{\mathsf{A}} &= \mathsf{CA}?\mathtt{x}\ ; \mathsf{AB}!\,(\mathtt{x} * 6) \\
L_{\mathsf{B}} &= ??? \\
L_{\mathsf{C}} &= ???
\end{aligned}
$$

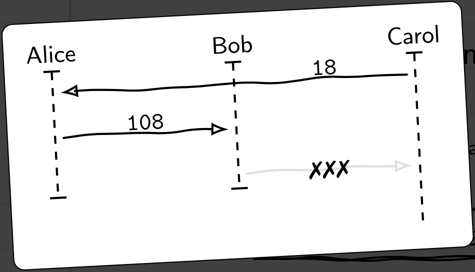
C ee

, POPL'13]

n a

anua

s de



1. Write global program

$$
\begin{aligned}
G = {} & \mathsf{C}.18 \rightarrow \mathsf{A}.\mathtt{x}\ ; \\
& \mathsf{A}.(\mathtt{x} * 6) \rightarrow \mathsf{B}.\mathtt{y}\ ; \\
& \textbf{if}\ \mathsf{B}.(\mathtt{y}\ \%\ 14 == 0) \\
& \quad \mathsf{B}.\texttt{"foo"} \rightarrow \mathsf{C}.\mathtt{z}
\end{aligned}
$$

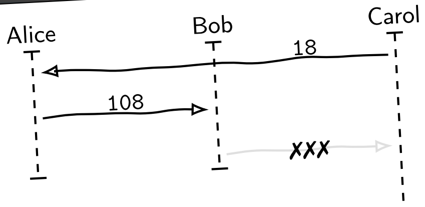
2. Dec… into local programs $L_1, \dots, L_n$ (automatically)

2. Decompose into local programs

$$
\begin{aligned}
L_{\mathsf{A}} &= \mathsf{CA}?\mathtt{x}\ ;\ \mathsf{AB}!\,(\mathtt{x} * 6) \\
L_{\mathsf{B}} &= \mathsf{AB}?\mathtt{y}\ ;\ \textbf{if}\ (\mathtt{y}\ \%\ 14 == 0)\ (\mathsf{BC}!\,\texttt{"foo"}) \\
L_{\mathsf{C}} &= ???
\end{aligned}
$$

ee

, POPL'13]



1. Write global program

$$\begin{aligned} G = {} & \mathsf{C}.18 \rightarrow \mathsf{A}.\mathtt{x} \,; \\ & \mathsf{A}.(\mathtt{x} * 6) \rightarrow \mathsf{B}.\mathtt{y} \,; \\ & \textbf{if } \mathsf{B}.(\mathtt{y} \,\%\, 14 == 0) \\ & \quad \mathsf{B}.\mathtt{"foo"} \rightarrow \mathsf{C}.\mathtt{z} \end{aligned}$$

2. Dec… into local programs $L_1, \ldots, L_n$ (automatically)

2. Decompose into local programs

$$L_{\mathsf{A}} = \mathsf{CA}?\mathtt{x} \,;\, \mathsf{AB}!\,(\mathtt{x} * 6)$$

$$L_{\mathsf{B}} = \mathsf{AB}?\mathtt{y} \,;\, \textbf{if } (\mathtt{y} \,\%\, 14 == 0)\ (\mathsf{BC}!\,\mathtt{"foo"})$$

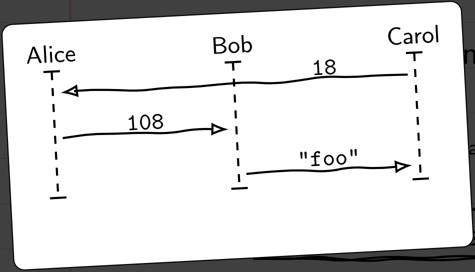
$$L_{\mathsf{C}} = \textit{undefined}$$

, POPL'13]

n a

anua

s de



1. Write global program

$$
\begin{aligned}
G = {} & \mathsf{C}.18 \rightarrow \mathsf{A}.\mathtt{x}\ ; \\
& \mathsf{A}.(\mathtt{x} * 6) \rightarrow \mathsf{B}.\mathtt{y}\ ; \\
& \sout{\textbf{if } \mathsf{B}.(\mathtt{y}\ \%\ 14 == 0)} \\
& \quad \mathsf{B}.\texttt{"foo"} \rightarrow \mathsf{C}.\mathtt{z}
\end{aligned}
$$

2. Dec into local programs $L_1, \ldots, L_n$ (automatically)

2. Decompose into local programs

$$L_{\mathsf{A}} = \mathsf{CA}?\mathtt{x}\ ; \mathsf{AB}!(\mathtt{x} * 6)$$

$$L_{\mathsf{B}} = \mathsf{AB}?\mathtt{y}\ ; \sout{\textbf{if } (\mathtt{y}\ \%\ 14 == 0)\ (}\mathsf{BC}!\texttt{"foo"}\sout{)}$$

$$L_{\mathsf{C}} = \sout{\textit{undefined}}\ \mathsf{CA}!18\ ; \mathsf{BC}?\mathtt{z}$$

ee

, POPL'13]

Question: Can we construct a functionally-correct and deadlock-free program for, e.g., *leader election*?

Answer: ???

Question: Can we construct a functionally-correct and deadlock-free program for, e.g., *leader election*?

Answer: No

Question: Can we construct a functionally-correct and deadlock-free program for, e.g., *leader election*?

Answer: No

Question: Why not?

Answer: ???

Question: Can we construct a functionally-correct and deadlock-free program for, e.g., *leader election*?

Answer: No

Question: Why not?

Answer: (1) No theory of functional correctness; (2) No theory of deadlock freedom of “multiparty conditions”

Question: Can we construct a functionally-correct and deadlock-free program for, e.g., *leader election*?

Answer: ~~No~~ Yes

Contributions – What?

(1) Functional correctness

(2) Deadlock fr. + multiparty conditions

Question: ~~Why not?~~

Answer: ~~(1) No theory of functional correctness; (2) No theory of deadlock freedom of “multiparty conditions”~~

Question: Can we construct a functionally-correct and deadlock-free program for, e.g., *leader election*?

Answer: ~~No~~ Yes

Question: ~~Why not?~~

Answer: ~~(1) No theory of functional correctness; (2) No theory of deadlock freedom of "multiparty conditions"~~

Contributions – What?

(1) Functional correctness

(2) Deadlock fr. + multiparty conditions

Contributions – How?

A *predicate transformer* for global programs

Functional correctness: If the precondition is true before executing, then the postcondition is true after

Deadlock freedom: Always, reduce or terminate

Functional correctness: If the precondition is true before executing, then the postcondition is true after

Deadlock freedom: Always, reduce or terminate

- One-party condition: (centralised; existing)

$$\begin{aligned}&\textbf{if}\ p.e\ (p.\mathtt{true} \rightarrowtriangle q_1.x_1\ ;\ \ldots\ ; p.\mathtt{true} \rightarrowtriangle q_n.x_n\ ; G_{\mathtt{true}})\\ &\qquad\ (p.\mathtt{false} \rightarrowtriangle q_1.x_1\ ;\ \ldots\ ; p.\mathtt{false} \rightarrowtriangle q_n.x_n\ ; G_{\mathtt{false}})\end{aligned}$$

Functional correctness: If the precondition is true before executing, then the postcondition is true after

Deadlock freedom: Always, reduce or terminate

- One-party condition: (centralised; existing)

$$\textbf{if } p.e\ (p.\texttt{true} \rightarrow q_1.x_1\ ;\ \ldots\ ;\ p.\texttt{true} \rightarrow$$

$$(p.\texttt{false} \rightarrow q_1.x_1\ ;\ \ldots\ ;\ p.\texttt{false} \rightarrow q_n.x_n\ ;\ G_{\texttt{false}})$$

Needed:
One-to-all
communications
("easy" to check)

Functional correctness: If the precondition is true before executing, then the postcondition is true after

Deadlock freedom: Always, reduce or terminate

- One-party condition: (centralised; existing)

$$\textbf{if}\ p.e\ (p.\texttt{true} \to q_1.x_1\ ;\ \ldots\ ;\ p.\texttt{true} \to$$

$$(p.\texttt{false} \to q_1.x_1\ ;\ \ldots\ ;\ p.\texttt{false} \to q_n.x_n\ ;\ G_{\texttt{false}})$$

Needed:
One-to-all
communications
("easy" to check)

- Multiparty condition: (decentralised; new)

$$\textbf{if}\ (r_1.e_1 \wedge \cdots \wedge r_n.e_n)\ G_{\texttt{true}}\ G_{\texttt{false}}$$

Functional correctness: If the precondition is true before executing, then the postcondition is true after

Deadlock freedom: Always, reduce or terminate

- One-party condition: (centralised; existing)

$$\textbf{if } p.e\ (p.\mathtt{true} \rightarrow q_1.x_1\ ;\ \ldots\ ;\ p.\mathtt{true} \rightarrow$$

$$(p.\mathtt{false} \rightarrow q_1.x_1\ ;\ \ldots\ ;\ p.\mathtt{false} \rightarrow q_n.x_n\ ;\ G_{\mathtt{false}})$$

Needed:
One-to-all
communications
("easy" to check)

- Multiparty condition: (decentralised; new)

$$\textbf{if } (r_1.e_1 \wedge \cdots \wedge r_n.e_n)\ G_{\mathtt{true}}$$

Needed:
Unanimity among all
("hard" to check)

Functional correctness: If the precondition is true before executing, then the postcondition is true after

Deadlock freedom: Always, reduce or terminate

- One-party condition: (centralised; existing)

$$\textbf{if } p.e\,(p.\mathtt{true} \rightarrowtriangle q_1.x_1 \,;\, \ldots \,;\, p.\mathtt{true} \rightarrowtriangle$$
$$(p.\mathtt{false} \rightarrowtriangle q_1.x_1 \,;\, \ldots \,;\, p.\mathtt{false} \rightarrowtriangle q_n.x_n \,;\, G_{\mathtt{false}})$$

Needed:
One-to-all
communications
("easy" to check)

- Multiparty condition: (decentralised; new)

$$\textbf{if } (r_1.e_1 \wedge \cdots \wedge r_n.e_n)\; G_{\mathtt{true}}$$

Needed:
Unanimity among all
("hard" to check)

Functional correctness: If the precondition is true before executing, then the postcondition is true after

$$\begin{aligned}G = {} & \mathsf{A}.\mathtt{x} \rightarrow \mathsf{B}.\mathtt{x}\ ;\\ & \mathsf{B}.\mathtt{y} \rightarrow \mathsf{A}.\mathtt{y}\ ;\\ & \textbf{if}\ (\mathsf{A}.(\mathtt{x} == \mathtt{y}) \wedge \mathsf{B}.(\mathtt{x}\ != \mathtt{y}))\\ & \quad \mathsf{B}.\mathtt{"foo"} \rightarrow \mathsf{A}.\mathtt{z}\\ & \quad \mathsf{A}.\mathtt{"bar"} \rightarrow \mathsf{B}.\mathtt{z}\end{aligned}$$

Deadlock ate

– One-par

Needed: ne-to-all munications sy" to check)

$$\textbf{if}\ p.e\ (p$$

$$(p.\mathtt{false} \rightarrow q_1.x_1\ ;\ \ldots\ ;\ p.\mathtt{false} \rightarrow q_n.x_n\ ;\ G_{\mathtt{false}})$$

– Multiparty condition: (decentralised; new)

$$\textbf{if}\ (r_1.e_1 \wedge \cdots \wedge r_n.e_n)\ G_{\mathtt{true}}$$

Needed:
Unanimity among all
("hard" to check)

$$\textit{This paper: } \text{``Kill } \underbrace{\text{two birds}}_{\substack{\text{functional correctness}\\ \text{and deadlock freedom}\\ \text{+ multiparty conditions}}} \text{ with } \overbrace{\text{one stone}}^{\text{predicate transformer}}\text{''}$$

predicate transformer

This paper: "Kill two birds with one stone"

functional correctness
and deadlock freedom
+ multiparty conditions

In three bullets:

- Idea goes back to Dijkstra in the 1970s (weakest preconditions)

This paper: “Kill $\underbrace{\text{two birds}}_{\substack{\text{functional correctness}\\ \text{and deadlock freedom}\\ \text{+ multiparty conditions}}}$ with $\overbrace{\text{one stone}}^{\text{predicate transformer}}$”

In three bullets:

- Idea goes back to Dijkstra in the 1970s (weakest preconditions)

- If G and χ are a global program and a postconditition, then $\phi(G, \chi)$ is a corresponding precondition

predicate transformer

This paper: "Kill two birds with one stone"

functional correctness
and deadlock freedom
+ multiparty conditions

In three bullets:

- Idea goes back to Dijkstra in the 1970s (weakest preconditions)
- If G and χ are a global program and a postconditition, then $\phi(G, \chi)$ is a corresponding precondition
- E.g.:

predicate transformer

This paper: “Kill two birds with one stone”

functional correctness
and deadlock freedom
+ multiparty conditions

In three bullets:

- Idea goes back to Dijkstra in the 1970s (weakest preconditions)

- If G and χ are a global program and a postcondititition, then $\phi(G, \chi)$ is a corresponding precondition

- E.g.: C.18 → A.x ; A.(x ∗ 6) → B.y

predicate transformer

This paper: "Kill two birds with one stone"

functional correctness
and deadlock freedom
+ multiparty conditions

In three bullets:

- Idea goes back to Dijkstra in the 1970s (weakest preconditions)
- If G and χ are a global program and a postconditition, then $\phi(G, \chi)$ is a corresponding precondition
- E.g.: C.18 → A.x ; A.(x ∗ 6) → B.y   B.y % 14 == 0

predicate transformer

This paper: "Kill two birds with one stone"

functional correctness
and deadlock freedom
+ multiparty conditions

In three bullets:

- Idea goes back to Dijkstra in the 1970s (weakest preconditions)
- If G and χ are a global program and a postconditition, then $\phi(G, \chi)$ is a corresponding precondition
- E.g.: $\phi(\, \mathrm{C.18} \rightarrow \mathrm{A.x}\, ;\, \mathrm{A.(x * 6)} \rightarrow \mathrm{B.y}\, ,\, \mathrm{B.y \,\%\, 14 == 0}\,) \equiv \perp$

predicate transformer

This paper: "Kill two birds with one stone"

functional correctness
and deadlock freedom
+ multiparty conditions

In three bullets:

- Idea goes back to Dijkstra in the 1970s (weakest preconditions)
- If G and χ are a global program and a postcondititition, then $\phi(G, \chi)$ is a corresponding precondition
- E.g.: $\phi(\,\mathrm{C.18} \rightarrow \mathrm{A.x}\,;\, \mathrm{A.(x * 6)} \rightarrow \mathrm{B.y}\,,\ \mathrm{B.y}\,\%\,12 == 0\,) \equiv \top$

Suppose we have syntax, semantics, “well-formedness”, and predicate transformer…

Main results

Suppose we have syntax, semantics, "well-formedness", and predicate transformer…

Main results

Theorem: If G is well-formed and $\phi(G, \chi) \equiv \top$, then G is functionally correct and deadlock-free (+ multiparty conditions)

Suppose we have syntax, semantics, "well-formedness", and predicate transformer...

Main results

Theorem: If G is well-formed and $\phi(G, \chi) \equiv \top$, then G is functionally correct and deadlock-free (+ multiparty conditions)

Theorem: If G is well-formed and $\phi(G, \chi) \equiv \top$, then $G \approx L_1 | \cdots | L_n$

Suppose we have syntax, semantics, "well-formedness", and predicate transformer...

Main results

Theorem: If G is well-formed and $\phi(G,\chi) \equiv \top$, then G is functionally correct and deadlock-free (+ multiparty conditions)

Theorem: If G is well-formed and $\phi(G,\chi) \equiv \top$, then $G \approx L_1 | \cdots | L_n$

Corollary: If G is well-formed and $\phi(G,\chi) \equiv \top$, then $L_1 | \cdots | L_n$ is functionally correct and deadlock-free (+ multiparty conditions)

The details...

Syntax:

$$
\begin{array}{rcl}
G & ::= & q.y := e \\
 & \mid & p.e \rightarrowtriangle q.y \\
 & \mid & G_1 \, ; G_2 \\
 & \mid & G_1 \parallel G_2 \\
 & \mid & \textbf{if } \bigwedge\{e_r\}_{r \in R} \, G_1 \, G_2 \\
 & \mid & \textbf{while } \bigwedge\{e_r\}_{r \in R} \, \{\psi_{\text{inv}}\} \, G
\end{array}
$$

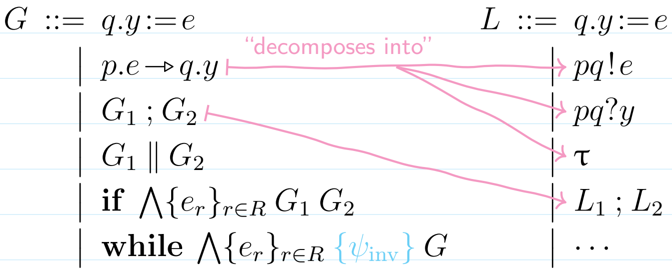
The details...

Syntax:

$$
\begin{array}{rcl}
G & ::= & q.y := e \\
 & \mid & p.e \rightarrowtriangle q.y \\
 & \mid & G_1 \,;\, G_2 \\
 & \mid & G_1 \parallel G_2 \\
 & \mid & \textbf{if } \bigwedge\{e_r\}_{r\in R}\, G_1\, G_2 \\
 & \mid & \textbf{while } \bigwedge\{e_r\}_{r\in R}\, \{\psi_{\text{inv}}\}\, G
\end{array}
\qquad\qquad
\begin{array}{rcl}
L & ::= & q.y := e \\
 & \mid & pq!e \\
 & \mid & pq?y \\
 & \mid & \boldsymbol{\tau} \\
 & \mid & L_1 \,;\, L_2 \\
 & \mid & \ldots
\end{array}
$$

The details...

Syntax:



$$
\begin{array}{rcl}
G & ::= & q.y := e \\
  & \mid & p.e \rightarrowtriangle q.y \\
  & \mid & G_1 \mathbin{;} G_2 \\
  & \mid & G_1 \parallel G_2 \\
  & \mid & \textbf{if } \bigwedge\{e_r\}_{r\in R}\, G_1\, G_2 \\
  & \mid & \textbf{while } \bigwedge\{e_r\}_{r\in R}\, \{\psi_{\mathrm{inv}}\}\, G
\end{array}
\qquad
\begin{array}{rcl}
L & ::= & q.y := e \\
  & \mid & pq\,!\,e \\
  & \mid & pq\,?\,y \\
  & \mid & \tau \\
  & \mid & L_1 \mathbin{;} L_2 \\
  & \mid & \dots
\end{array}
$$

The details...

Semantics:

- Abstract reductions: (symbolic)

$$G \xrightarrow{\psi,\gamma} G' \qquad L \xrightarrow{\psi,\lambda} L' \qquad L_1|\cdots|L_n \xrightarrow{\psi,\gamma} L'_1|\cdots|L'_n$$

The details...

Semantics:

- Abstract reductions: (symbolic)

$$G \xrightarrow{\psi,\gamma} G' \qquad L \xrightarrow{\psi,\lambda} L' \qquad L_1|\cdots|L_n \xrightarrow{\psi,\gamma} L_1'|\cdots|L_n'$$

- Concrete reductions: (explicit)

$$(G,\mathcal{S}) \xrightarrow{\gamma} (G',\mathcal{S}') \quad (L_1|\cdots|L_n,\mathcal{S}) \xrightarrow{\gamma} (L_1'|\cdots|L_n',\mathcal{S}')$$

The details...

Semantics:

- Abstract reductions: (symbolic)

$$G \xrightarrow{\psi,\gamma} G' \qquad L \xrightarrow{\psi,\lambda} L' \qquad L_1|$$

Sequencing is *weak*:

$$\frac{G_1 \cap \gamma = \emptyset \quad G_2 \xrightarrow{\psi,\gamma} G_2'}{G_1 \,;\, G_2 \xrightarrow{\psi,\gamma} G_1 \,;\, G_2'}$$

[Rensink & Wehrheim, CONCUR'94]

- Concrete reductions: (explicit)

$$(G, \mathcal{S}) \xrightarrow{\gamma} (G', \mathcal{S}') \quad (L_1|\cdots|L_n, \mathcal{S}) \xrightarrow{\gamma} (L_1'|\cdots|L_n', \mathcal{S}')$$

The details...

Well-formedness:

- In $G_1 \parallel G_2$, the channels that occur in G_1 are disjoint from those that occur in G_2
- In **if** $\bigwedge\{e_r\}_{r\in R}\, G_1\, G_2$ and **while** $\bigwedge\{e_r\}_{r\in R}\, \{\psi_{\mathrm{inv}}\}\, G$, every process has a conjunct (multiparty conditions are “total”)

The details...

Predicate transformer: (excerpt)

$$
\begin{aligned}
\phi(q.y := e, \chi) &= \chi[e/q.y] \\
\phi(p.e \rightarrowtriangle q.y, \chi) &= \chi[e/q.y] \\
\phi(G_1 \,;\, G_2, \chi) &= \phi(G_1, \phi(G_2, \chi)) \\
\phi(G_1 \parallel G_2, \chi) &= \\
&\hspace{-6em}\begin{cases} \phi(G_1, \phi(G_2, \chi)) & \text{if } \mathsf{var}(G_1) \cap \mathsf{var}(G_2) = \emptyset \\ \texttt{false} & \text{otherwise} \end{cases}
\end{aligned}
$$

The details...

Predicate transformer: (excerpt)

$$
\begin{aligned}
\phi(q.y := e, \chi) &= \chi[e/q.y] \\
\phi(p.e \rightarrowtriangle q.y, \chi) &= \chi[e/q.y] \\
\phi(G_1 \,;\, G_2, \chi) &= \phi(G_1, \phi(G_2, \chi)) \\
\phi(G_1 \parallel G_2, \chi) &= \\
&\begin{cases} \phi(G_1, \phi(G_2, \chi)) & \text{if } \mathsf{var}(G_1) \cap \mathsf{var}(G_2) = \emptyset \\ \texttt{false} & \text{otherwise} \end{cases}
\end{aligned}
$$

← Surprisingly complicated case

The details...

Predicate transformer: (excerpt)

$$
\begin{aligned}
&\phi(\textbf{if } \bigwedge\{e_r\}_{r\in R}\, G_1\, G_2, \chi) = \\
&\quad (\bigwedge\{\ \ e_r\}_{r\in R} \Rightarrow \phi(G_1, \chi)) \wedge \\
&\quad (\bigwedge\{\neg e_r\}_{r\in R} \Rightarrow \phi(G_2, \chi)) \wedge \\
&\quad (\bigwedge\{e_{r_1} \Rightarrow e_{r_2}\}_{r_1, r_2\in R})
\end{aligned}
$$

The details...

Predicate transformer: (excerpt)

$$
\begin{aligned}
&\phi(\mathbf{if}\ \bigwedge\{e_r\}_{r\in R}\ G_1\ G_2, \chi) = \\
&\quad (\bigwedge\{\ \ e_r\}_{r\in R} \Rightarrow \phi(G_1, \chi)) \wedge \\
&\quad (\bigwedge\{\neg e_r\}_{r\in R} \Rightarrow \phi(G_2, \chi)) \wedge \\
&\quad (\bigwedge\{e_{r_1} \Rightarrow e_{r_2}\}_{r_1, r_2 \in R})
\end{aligned}
$$

⟵ Essential for deadlock freedom

The details...

Predicate transformer: (excerpt)

$$
\begin{aligned}
&\phi(\textbf{while } \bigwedge\{e_r\}_{r\in R}\ \{\psi_{\text{inv}}\}\ G, \chi) = \\
&\quad \psi_{\text{inv}} \wedge \forall(\psi_{\text{inv}} \Rightarrow (\bigwedge\{\ \ e_r\}_{r\in R} \Rightarrow \phi(G,\chi)) \wedge \\
&\qquad\qquad\qquad\qquad\quad (\bigwedge\{\neg e_r\}_{r\in R} \Rightarrow \chi) \wedge \\
&\qquad\qquad\qquad\qquad\quad (\bigwedge\{e_{r_1} \Rightarrow e_{r_2}\}_{r_1,r_2\in R}))
\end{aligned}
$$

The details...

Predicate transformer: (excerpt)

$$
\begin{aligned}
&\phi(\mathbf{while}\ \bigwedge\{e_r\}_{r\in R}\ \{\psi_{\text{inv}}\}\ G,\chi) = \\
&\quad \psi_{\text{inv}} \wedge \forall(\psi_{\text{inv}} \Rightarrow (\bigwedge\{\ \ e_r\}_{r\in R} \Rightarrow \phi(G,\chi)) \wedge \\
&\qquad\qquad\qquad\qquad (\bigwedge\{\neg e_r\}_{r\in R} \Rightarrow \chi) \wedge \\
&\qquad\qquad\qquad\qquad (\bigwedge\{e_{r_1} \Rightarrow e_{r_2}\}_{r_1,r_2\in R}))
\end{aligned}
$$

Essential for deadlock freedom ⟶

Suppose we have syntax, semantics, "well-formedness", and predicate transformer...

Main results

Theorem: If G is well-formed and $\phi(G, \chi) \equiv \top$, then G is functionally correct and deadlock-free (+ multiparty conditions)

Theorem: If G is well-formed and $\phi(G, \chi) \equiv \top$, then $G \approx L_1 | \cdots | L_n$

Corollary: If G is well-formed and $\phi(G, \chi) \equiv \top$, then $L_1 | \cdots | L_n$ is functionally correct and deadlock-free (+ multiparty conditions)

Question: Can we construct a functionally-correct and deadlock-free program for, e.g., *leader election*?

Answer: ~~No~~ Yes

Contributions – What?

(1) Functional correctness

(2) Deadlock fr. + multiparty conditions

Contributions – How?

A *predicate transformer* for global programs

Question: ~~Why not?~~

Answer: ~~(1) No theory of functional correctness; (2) No theory of deadlock freedom of "multiparty conditions"~~

Question: Can we construct a functionally-correct and deadlock-free program for, e.g., *leader election*?

Answer: ~~No~~ Yes

Question: ~~Why not~~?

Answer: ~~(1) No theory of functional correctness; (2) No theory of deadlock freedom of "multiparty conditions"~~

Contributions – What?

(1) Functional correctness

(2) Deadlock fr. + multiparty conditions

Contributions – How?

A *predicate transformer* for global programs

Question: Can we construct a functionally-correct and deadlock-free program for, e.g., *leader election*?

Answer: ~~No~~ Yes

Contributions – What?

(1) Functional correctness

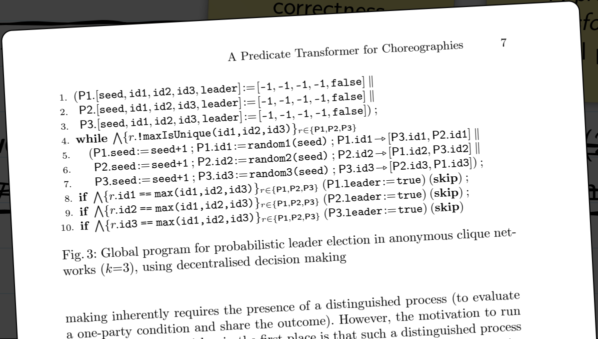
Contributions – How?

A *predicate* ...*former* for ...l programs

Ques

Answ ~~(2) No~~

~~theory~~ ...ns"



A Predicate Transformer for Choreographies 7

```
(P1.[seed, id1, id2, id3, leader] := [-1, -1, -1, -1, false] ||
 P2.[seed, id1, id2, id3, leader] := [-1, -1, -1, -1, false] ||
 P3.[seed, id1, id2, id3, leader] := [-1, -1, -1, -1, false]) ;
while ⋀{r.!maxIsUnique(id1,id2,id3)}_{r∈{P1,P2,P3}}
  (P1.seed := seed+1 ; P1.id1 := random1(seed) ; P1.id1 → [P3.id1, P2.id1] ||
   P2.seed := seed+1 ; P2.id2 := random2(seed) ; P2.id2 → [P1.id2, P3.id2] ||
   P3.seed := seed+1 ; P3.id3 := random3(seed) ; P3.id3 → [P2.id3, P1.id3]) ;
if ⋀{r.id1 == max(id1,id2,id3)}_{r∈{P1,P2,P3}} (P1.leader := true) (skip) ;
if ⋀{r.id2 == max(id1,id2,id3)}_{r∈{P1,P2,P3}} (P2.leader := true) (skip) ;
if ⋀{r.id3 == max(id1,id2,id3)}_{r∈{P1,P2,P3}} (P3.leader := true) (skip)
```

Fig. 3: Global program for probabilistic leader election in anonymous clique networks (k=3), using decentralised decision making

making inherently requires the presence of a distinguished process (to evaluate a one-party condition and share the outcome). However, the motivation to run ... the first place is that such a distinguished process

Question: Can we construct a functionally-correct and deadlock-free program for, e.g., *leader election*?

Answer: ~~No~~ Yes

Question: ~~Why not?~~

Answer: ~~(1) No theory of functional correctness; (2) No theory of deadlock freedom of "multiparty conditions"~~

Contributions – What?

(1) Functional correctness

(2) Deadlock fr. + multiparty conditions

Contributions – How?

A *predicate transformer* for global programs

Thank you (future work: asynchrony, and more)