Type-Based Analysis of Concurrent Programs

Naoki Kobayashi
University of Tokyo

Thanks to: A.Igarashi, B.Pierce, D.Turner

Many Concurrent Programming Languages Have Been Proposed

- Functional languages + concurrency primitives
  - CML[Reppy91], Concurrent Haskell,..
- Concurrent object-oriented languages
- CLP, CCP
- Process calculi-based languages
  - Pict[Pierce&Turner], HACL[Kobayashi&Yonezawa], Schematic[Taura&Yonezawa],..

Main Target

These languages may/may not be implemented on parallel machines.

What Are Types Useful For?

- Static detection of type mismatch
- Elimination of run-time type checking
- Tag-free garbage collection
- Type-directed marshalling/unmarshalling

Something more is necessary?

Difficulty in Compilation of Concurrent Programs

- More dynamic behavior than sequential programs: Compare
  - Function or method call in a sequential language:

    let x=f(v) in e
  - "Function" or method call in a concurrent language:

    new r in (*f!(v, r)* | *r?x.P*)

    - No guarantee of:

      the result being available when *r?x.P* is scheduled

      r being unnecessary after the value of *x* is extracted.

$\Rightarrow$ *Recover information on program behavior by static analysis!*

Traditional Type Systems Are Insufficient

- Evaluation of "functions" may get stuck
  - fun f(n) = let val ch = channel() in accept(ch)+n+1 end

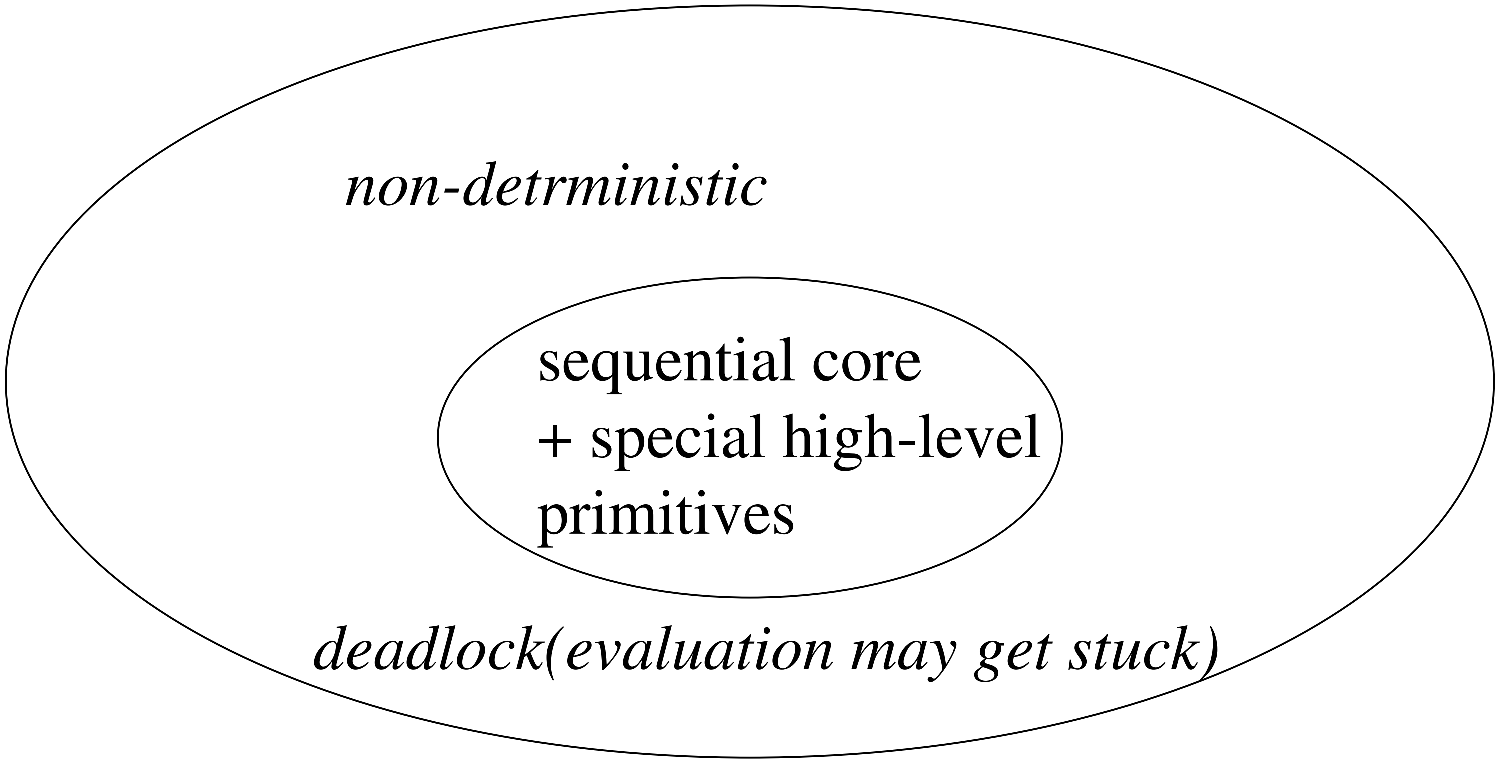
  val f = fn: int →int

- "Function" may return a non-deterministic result.
  - fun g(n) = let val ch=channel() in (spawn(fn ()=> send(ch,1)); spawn(fn ()=> send(ch,2)); accept(ch)+n) end

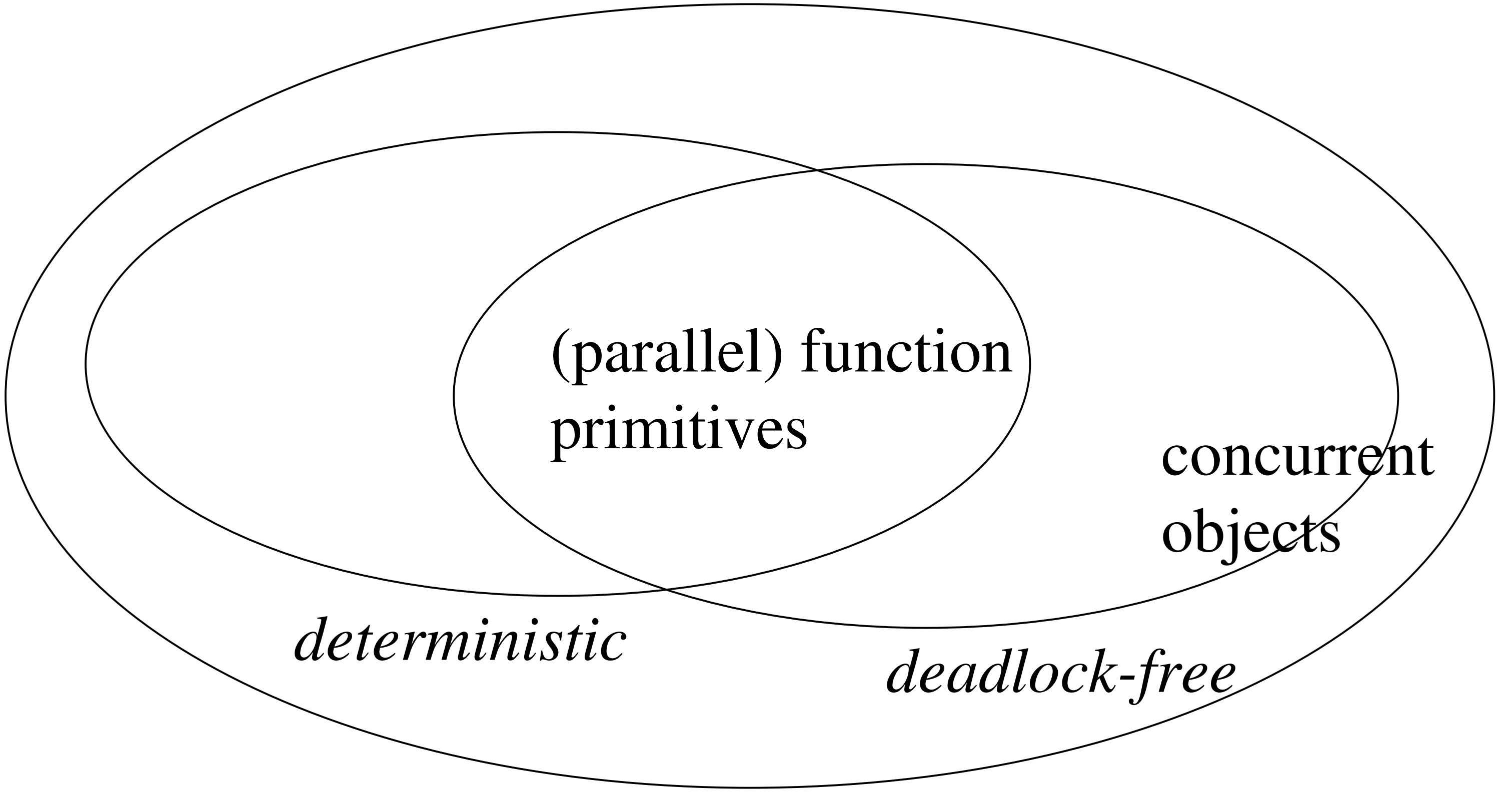
  val g = fn: int →int

Situations in Traditional "Typed" Concurrent Languages

♦ Complete disorder on the outside of the sequential core



Goal: Static Type Discipline to Control Program Behavior



- In "*new r in (f!(v,r)* | *r?x.P),*" we know
  - f returns a value to r
  - r is unnecessary after the value of x is read

Outline

- ***Overview***
  - Target Language
  - Why "Type-Based" Analysis (Why Not Abstract Interpretation, etc.)
  - What We Want to Analyze
  - Basic Ideas of Type-Based Analysis
  - How to Prove Correctness of Analysis and Optimization
- Linear Channel Type System
- Deadlock-Free Type System

Target Language

- Thread creation: $P \mid Q$
- First-class communication channels: *new x in P*
- Send/receive operations: $x!v$, $x?y.P$, $def\ x(y)=P\ in\ Q$

  $x!v \mid x?y.P \rightarrow [v/y]P$ (c.f. $(\lambda y.e)v \rightarrow [v/y]e$)

  $def\ x(y)=P\ in\ (x!v \mid Q) \rightarrow def\ x(y)=P\ in\ ([v/y]P \mid Q)$
- Sequential primitives (functions, objects, etc.) may/may not be included.

Why Type-Based Analysis?

- Abstract interpretation is difficult
  - An exponential number of execution paths

    $\alpha(P \mid Q) = \alpha(P) \mid \alpha(Q) = X \mid Y$

Why Type-Based Analysis?

- Abstract interpretation is difficult
  - An exponential number of execution paths

$$\alpha(P \mid Q) = \alpha(P) \mid \alpha(Q) = X \mid Y$$

$$X \mid Y$$

Why Type-Based Analysis?

- Abstract interpretation is difficult
  - An exponential number of execution paths

$$\alpha(P \mid Q) = \alpha(P) \mid \alpha(Q) = X \mid Y$$

$$X \mid Y \;\; \rightarrow Z \mid Y \rightarrow Z \mid W$$

Why Type-Based Analysis?

- Abstract interpretation is difficult
  - An exponential number of execution paths

$$\alpha(P \mid Q) = \alpha(P) \mid \alpha(Q) = X \mid Y$$

$$X \mid Y \quad \rightarrow X \mid W \rightarrow Z \mid W$$

Why Type-Based Analysis?

- Abstract interpretation is difficult
  - An exponential number of execution paths

$$\alpha(P \mid Q) = \alpha(P) \mid \alpha(Q) = X \mid Y$$

$$X \mid Y \quad \rightarrow U$$

Why Type-Based Analysis?

- Abstract interpretation is difficult
  - An exponential number of execution paths

$$\alpha(P \mid Q) = \alpha(P) \mid \alpha(Q) = X \mid Y$$

$$X \mid Y \quad \rightarrow U$$

(c.f.) Typing rules are compositional:

$$\frac{\Gamma \vdash P{:}\mathrm{ok} \quad \Delta \vdash Q{:}\mathrm{ok}}{\Gamma + \Delta \vdash P \mid Q{:}\ \mathrm{ok}}$$

Why Type-Based Analysis?

- Abstract interpretation is difficult
  - An exponential number of execution paths

    $$\alpha(P \mid Q) = \alpha(P) \mid \alpha(Q) = X \mid Y$$

    $$X \mid Y \to Z \mid Y,\ X \mid Y \to X \mid W,\ \ X \mid Y \to U$$

  - Change of program behavior caused by abstraction of communication channels

$$x!1 \mid y?n.\mathbf{0} \not\to$$

$$\Downarrow \quad \alpha(x) = \alpha(y) = c$$

$$c!1 \mid c?n.\mathbf{0} \quad \to \mathbf{0}$$

What Information Is Useful?

What Information Is Useful?

- Whether a process uses a channel for input or output [Pierce & Sangiorgi, LICS93]

What Information Is Useful?

- Whether a process uses a channel for input or output [Pierce & Sangiorgi, LICS93]
  - Program safety: prevent a message being stolen
  - Optimization: Improve dataflow information

What Information Is Useful?

- Whether a process uses a channel for input or output [Pierce and Sangiorgi, LICS93]
- How many times a communication channel is used [Kobayashi, Pierce, and Turner, POPL96]

What Information Is Useful?

- Whether a process uses a channel for input or output [Pierce and Sangiorgi, LICS93]
- How many times a communication channel is used [Kobayashi, Pierce, and Turner, POPL96]
  - Program safety: guarantee of a reply

What Information Is Useful?

- Whether a process uses a channel for input or output [Pierce and Sangiorgi, LICS93]
- How many times a communication channel is used [Kobayashi, Pierce, and Turner, POPL96]
  - Program safety: guarantee of a reply
  - Optimization: optimal channel representation, elimination of redundant communication

What Information Is Useful?

- Whether a process uses a channel for input or output [Pierce and Sangiorgi, LICS93]
- How many times a communication channel is used [Kobayashi, Pierce, and Turner, POPL96]
- How many processes try to communicate on a channel simultaneously [Kobayashi et al., SAS95]

What Information Is Useful?

- Whether a process uses a channel for input or output [Pierce and Sangiorgi, LICS93]
- How many times a communication channel is used [Kobayashi, Pierce, and Turner, POPL96]
- How many processes try to communicate on a channel simultaneously [Kobayashi et al., SAS95]
  - Elimination of unnecessary lock

What Information Is Useful?

- Whether a process uses a channel for input or output [Pierce and Sangiorgi, LICS93]
- How many times a communication channel is used [Kobayashi, Pierce, and Turner, POPL96]
- How many processes try to communicate on a channel simultaneously [Kobayashi et al., SAS95]
  - Elimination of unnecessary lock
  - Optimization of channel representation

What Information Is Useful?

- Whether a process uses a channel for input or output [Pierce and Sangiorgi, LICS93]
- How many times a communication channel is used [Kobayashi, Pierce, and Turner, POPL96]
- How many processes access a channel simultaneously [Kobayashi et al., SAS95]
- In which order communications take place [Kobayashi, LICS97]

What Information Is Useful?

- Whether a process uses a channel for input or output [Pierce and Sangiorgi, LICS93]
- How many times a communication channel is used [Kobayashi, Pierce, and Turner, POPL96]
- How many processes access a channel simultaneously [Kobayashi et al., SAS95]
- In which order communications take place [Kobayashi, LICS97]
  - Program safety: guarantee of deadlock-freedom
  - Optimization: optimal process scheduling, elimination of run-time channel status check

Basic Ideas of Type-Based Analysis

- Put extra information into types (c.f. effect systems)

Basic Ideas of Type-Based Analysis

- Put extra information into types (c.f. effect systems)

  Ch(τ) : Type of channels through which values of type τ can be sent/received

Basic Ideas of Type-Based Analysis

- Put extra information into types (c.f. effect systems)

$$\mathrm{Ch}(\tau, <I, O>)$$

Type of channels through which
values of type τ can be **sent *I* times**
and **received *O* times**

Basic Ideas of Type-Based Analysis

- Put extra information into types (c.f. effect systems)

  $\mathrm{Ch}(\tau, N)$

  Type of channels through which
  values of type τ can be sent/received
  simultaneously by at most N processes

Basic Ideas of Type-Based Analysis

- Put extra information into types (c.f. effect systems)

  Ch(τ, t)

  Type of channels through which values of type τ can be sent/received **at time t**

Basic Ideas of Type-Based Analysis

- Put extra information into types (c.f. effect systems)

- Construct type check rules

  ⇒algorithm for verification of programmers' declaration

- Construct type reconstruction rules

  ⇒algorithm for automatic analysis

How to Prove Correctness of Analysis?

- Define *typed* reduction relation: $(\Gamma, P) \rightarrow (\Delta, Q)$
  - E.g. $(x{:}\mathrm{Ch}(\mathrm{Int}, \langle \mathrm{I}, \mathrm{O} \rangle),\ x!1 \mid x?n.P) \rightarrow$
    $(x{:}\mathrm{Ch}(\mathrm{Int}, \langle \mathrm{I}-1, \mathrm{O}-1 \rangle),\ [1/n]P)$
- Prove lack of immediate violation of type information
  - E.g. $x{:}\mathrm{Ch}(\mathrm{Int}, \langle 0, 0 \rangle) \not\vdash x!1$
- Prove subject reduction
  - If $\Gamma \vdash P$ and $(\Gamma, P) \rightarrow (\Delta, Q)$, then $\Delta \vdash Q$

Justification of Optimization: Typed Process Equivalence

- Standard Process Equivalence:
  - $P \approx Q$ *iff* $C[P]$ and $C[Q]$ exhibit the same communication behavior for any context $C[\ ]$
- Typed Process Equivalence [Pierce&Sangiorgi, LICS93&POPL97; Kobayashi, Pierce&Turner POPL96]:
  - $P \approx Q$ *iff* $C[P]$ and $C[Q]$ exhibit the same communication behavior for any context $C[\ ]$ such that $C[P]$ and $C[Q]$ are well typed

E.g. In general,

$$x!y \not\approx x!z$$

but if a value received through x must not be used,

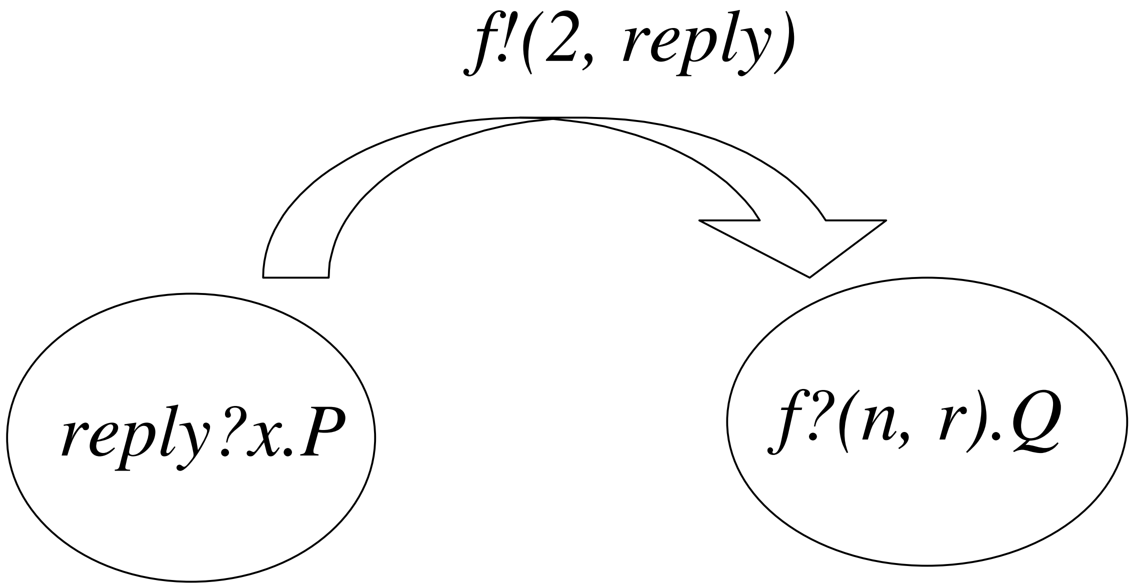
$$x!y \approx x!z$$

Outline:

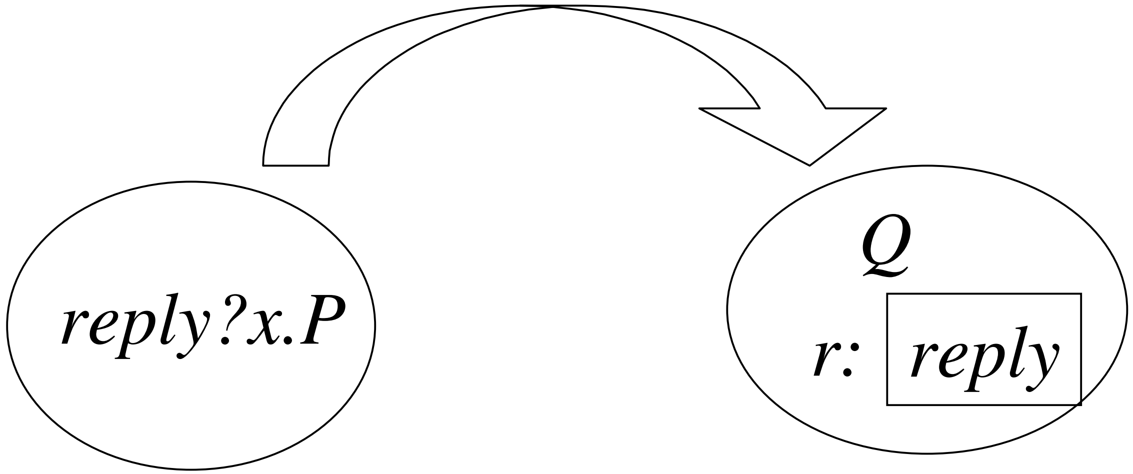
- Overview
- ***Linear (Use-Once) Channel Type System*** [Kobayashi, Pierce&Turner, POPL96]
  - *What for?*
  - *Types & type derivation*
  - *Type reconstruction* [Igarashi&Kobayashi,SAS97]
  - *Results of experiments*
- Deadlock-Free Type System

Linear (Use-Once) Channel: Why Is It Useful?

- From programmers' point of view
  - Program safety: Prevent a request from being ignored



Send a request to the channel "*f*"



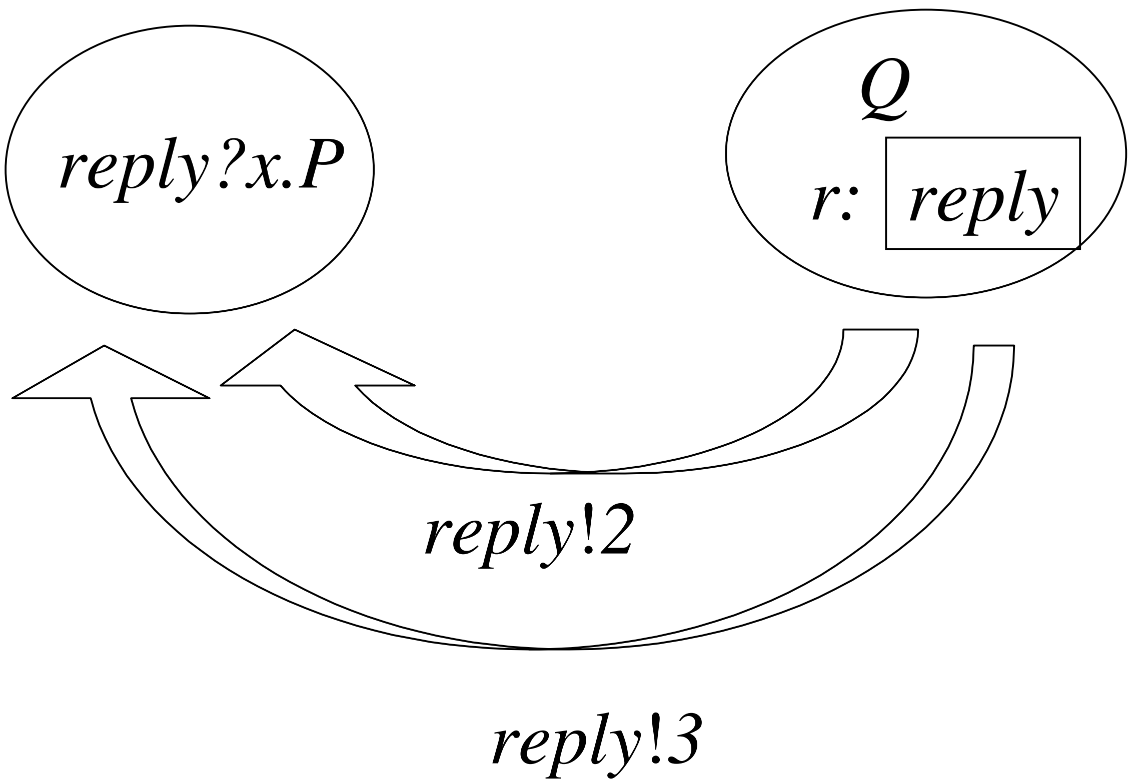
The request has been accepted

Without Linear Channel Types:

A result may not be returned

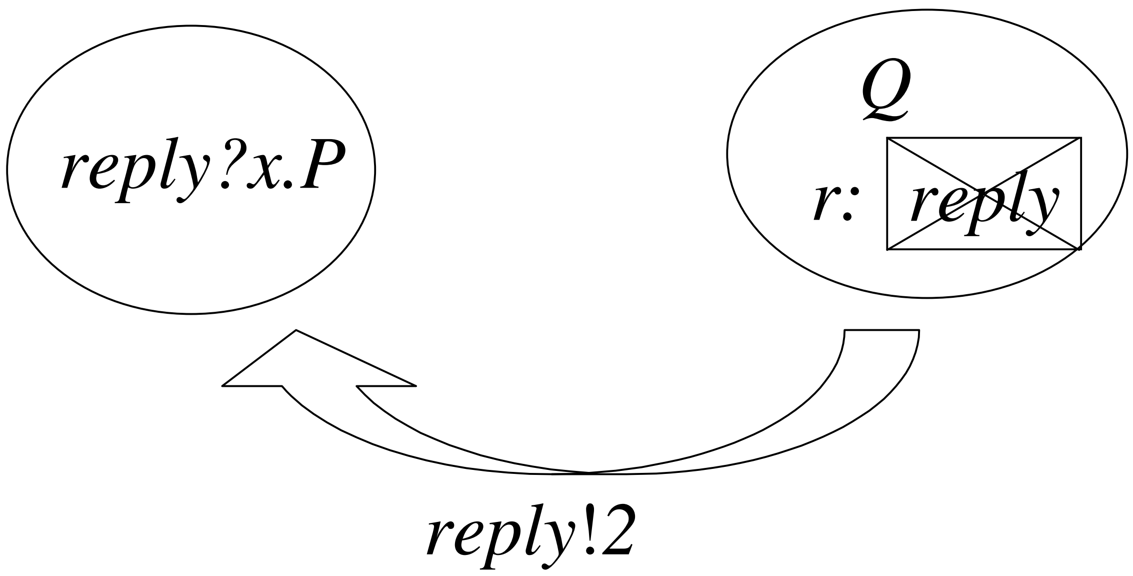
Without Linear Channel Types:

Multiple results may be returned!



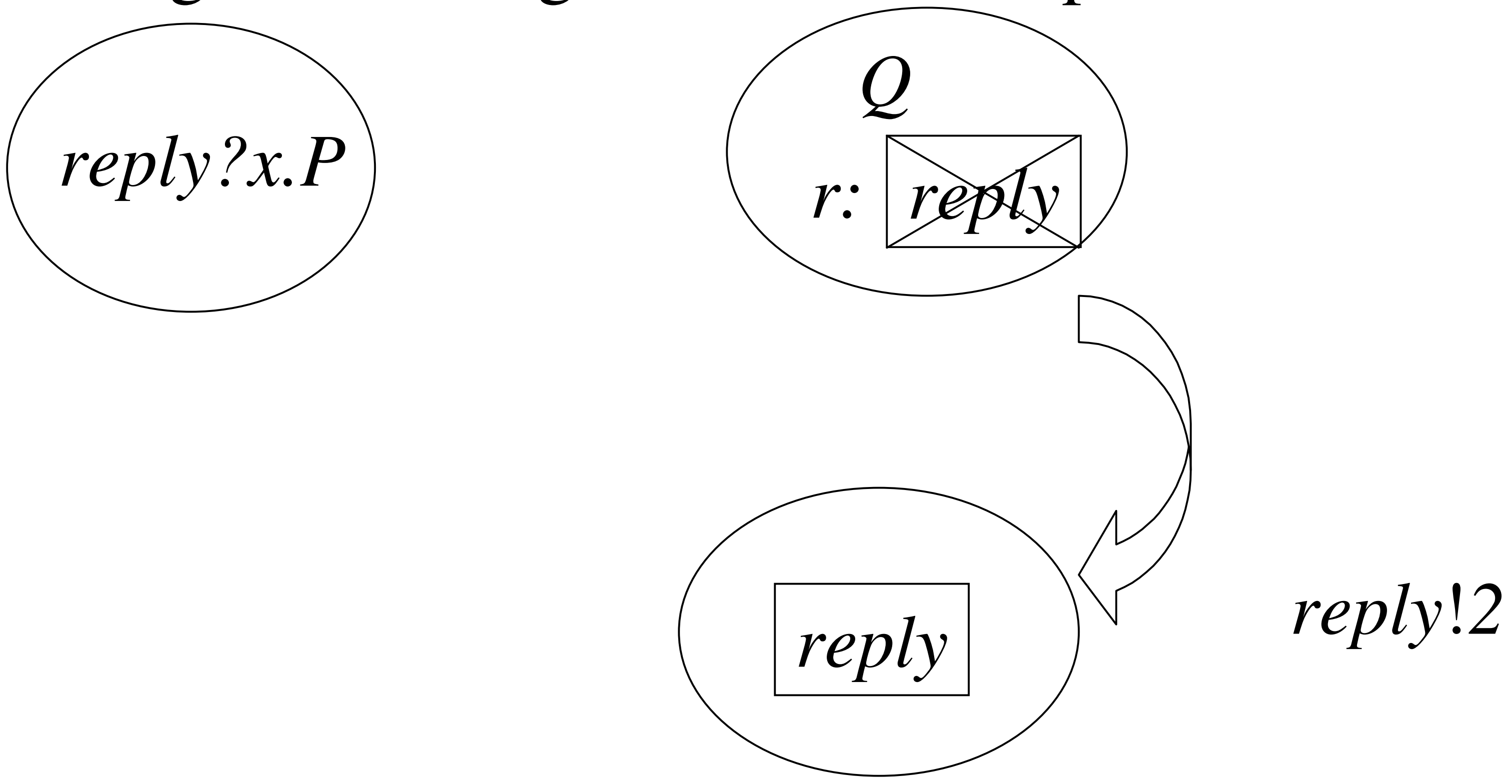
With Linear Channel Types:

The receiver must return a result, or ..



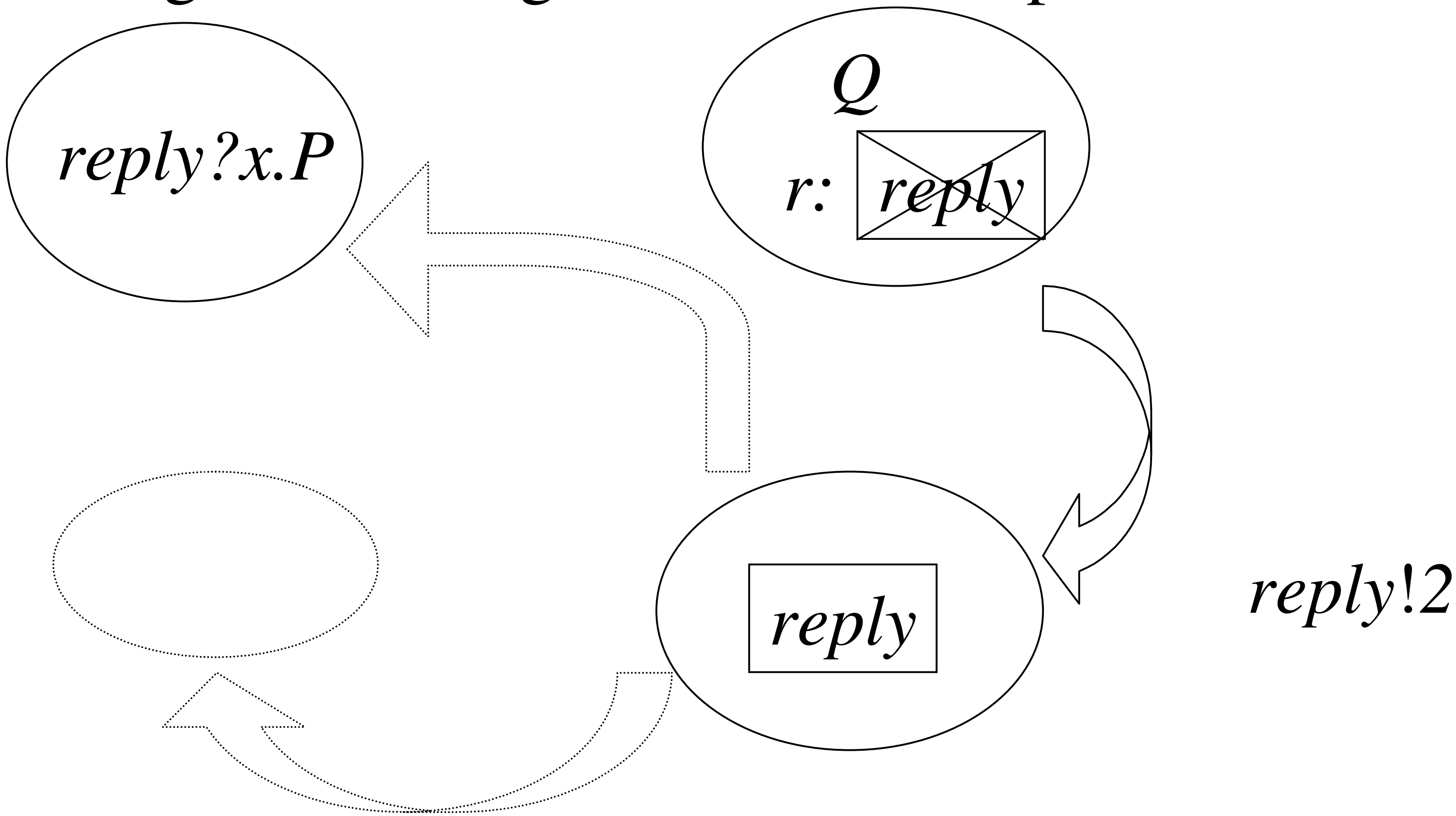
With Linear Channel Types:

The receiver must return a result, or
must delegate the obligation to another process



With Linear Channel Types:

The receiver must return a result, or
must delegate the obligation to another process



Linear (Use-Once) Channel: Why Is It Useful?

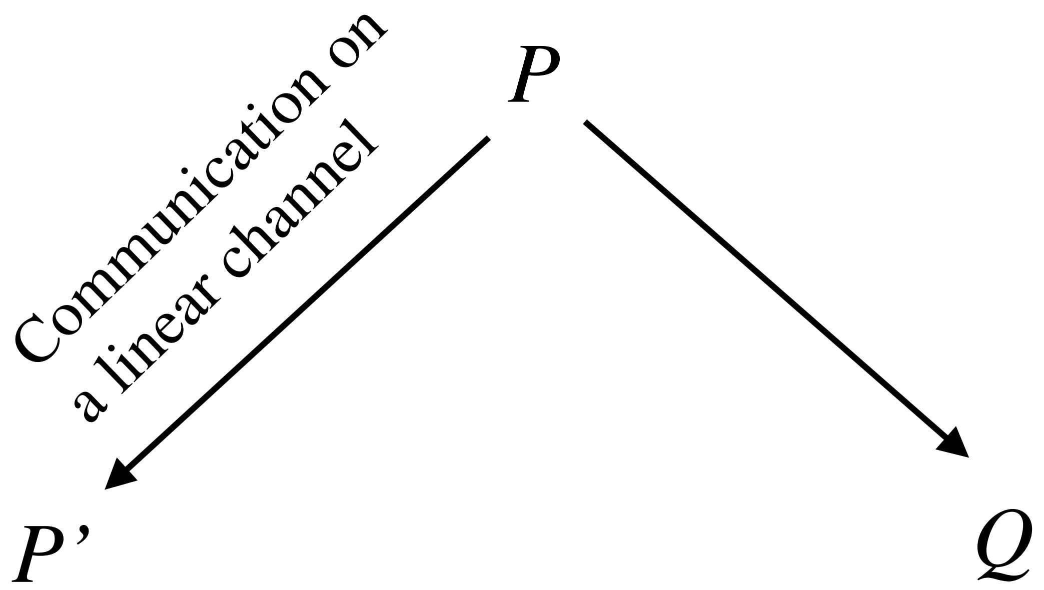
- From programmers' point of view
  - Program safety: Prevent a request being ignored
  - Help reasoning: Make the order of communications unimportant (partial confluence)

Reduction on Linear Channel Is Confluent

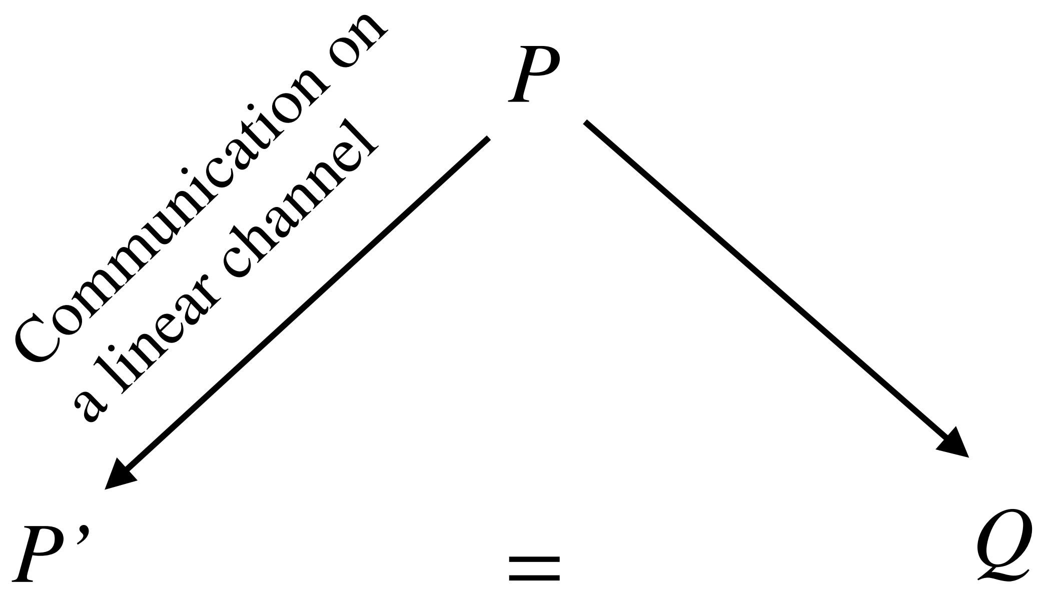
Reduction on Linear Channel Is Confluent

$$\begin{tikzcd} & P \arrow[dl, "\text{Communication on a linear channel}"'] \\ P' & \end{tikzcd}$$

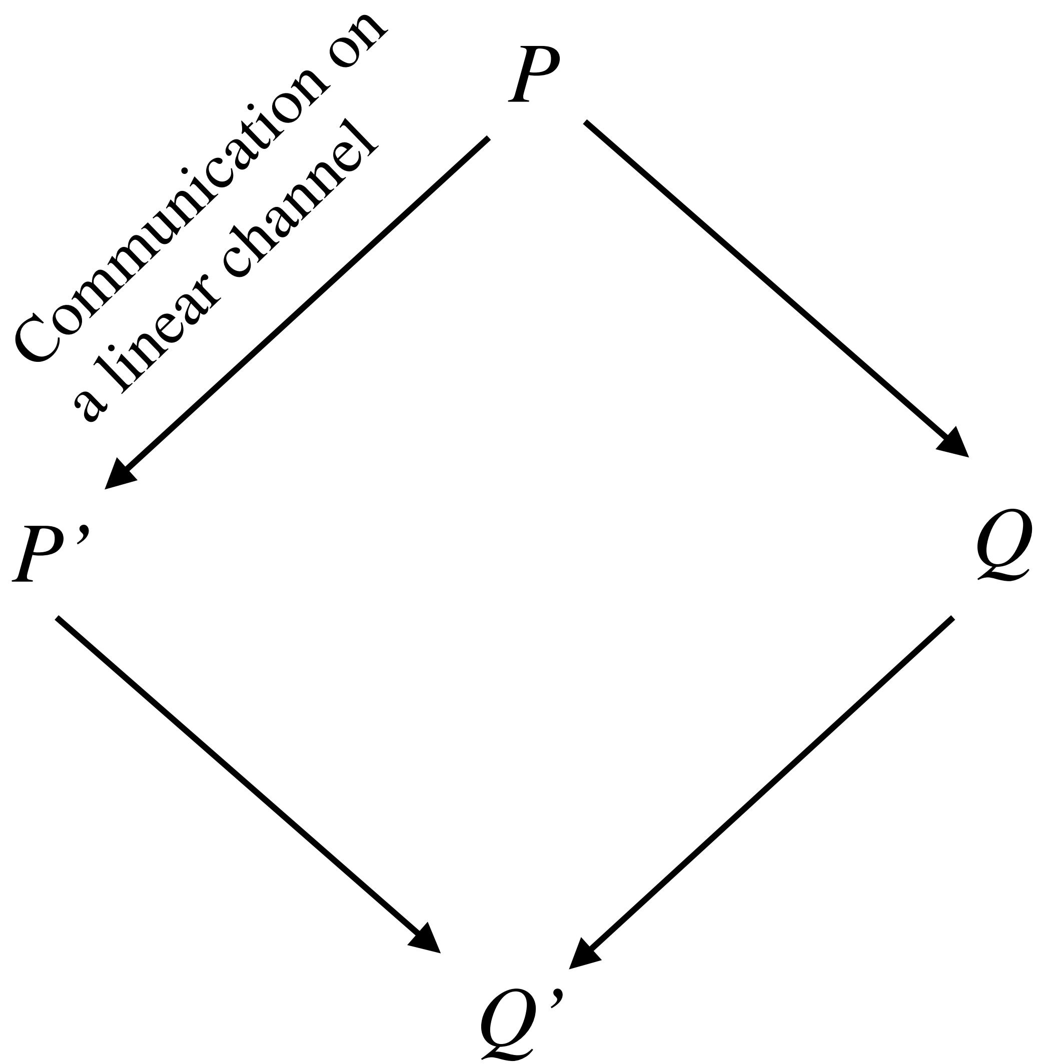
Reduction on Linear Channel Is Confluent



Reduction on Linear Channel Is Confluent



Reduction on Linear Channel Is Confluent



Linear (Use-Once) Channel:
Why Is It Useful?

Linear (Use-Once) Channel: Why Is It Useful?

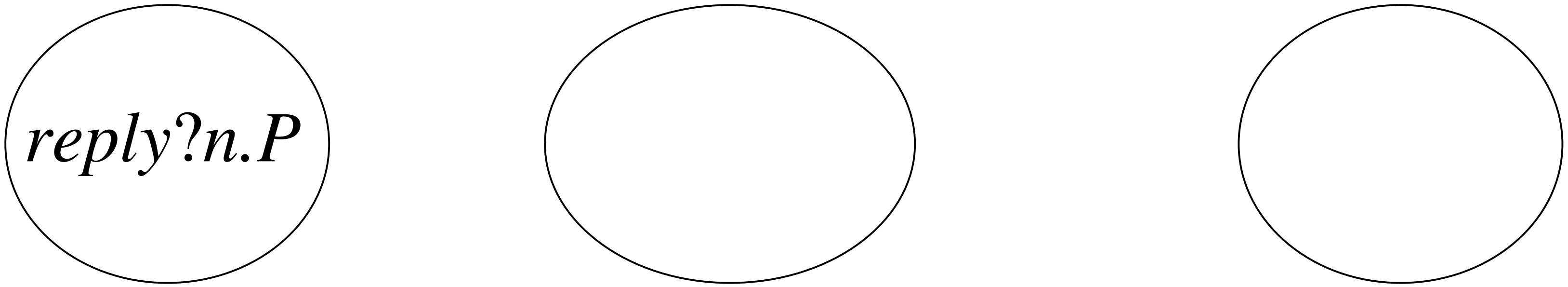
- From programmers' point of view
  - Program safety: Prevent a request being ignored
  - Help reasoning: Make the order of communications irrelevant (partial confluence)

Linear (Use-Once) Channel: Why Is It Useful?

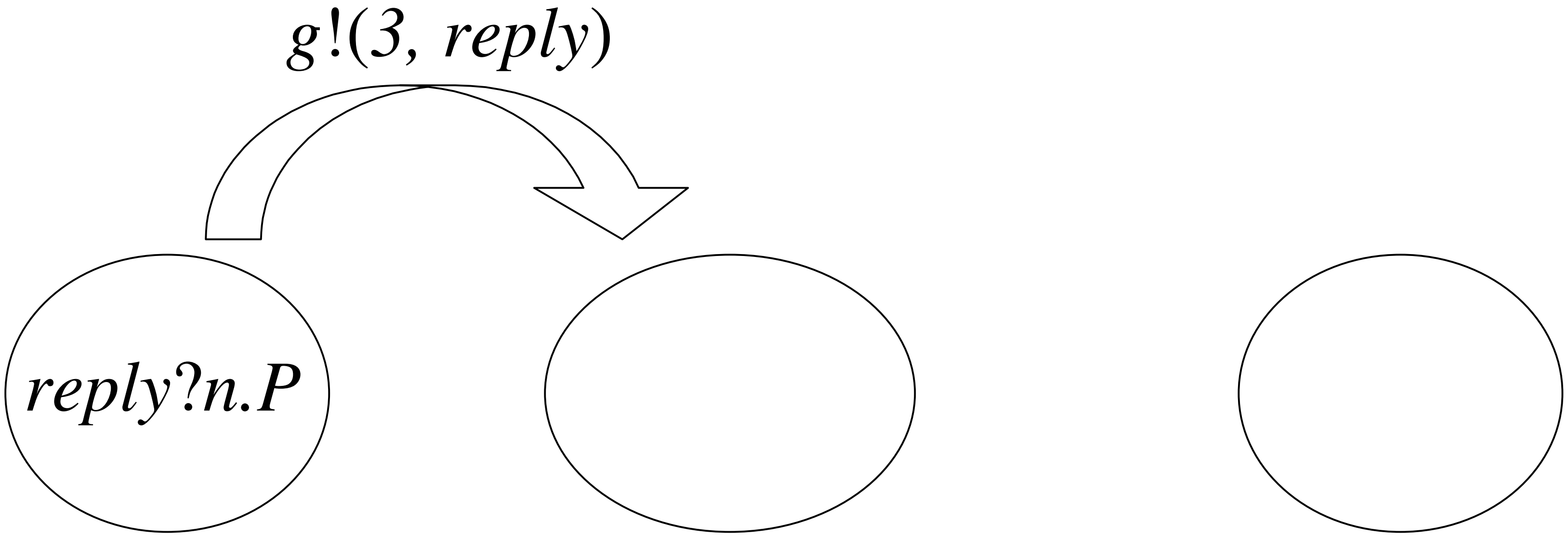
- From programmers' point of view
  - Program safety: Prevent a request being ignored
  - Help reasoning: Make the order of communications irrelevant (partial confluence)
- From compilers' point of view
  - Elimination of redundant communication

$$new\ s.(f![v, s] \mid s?[w].r![w]) \approx f!(v, r)$$

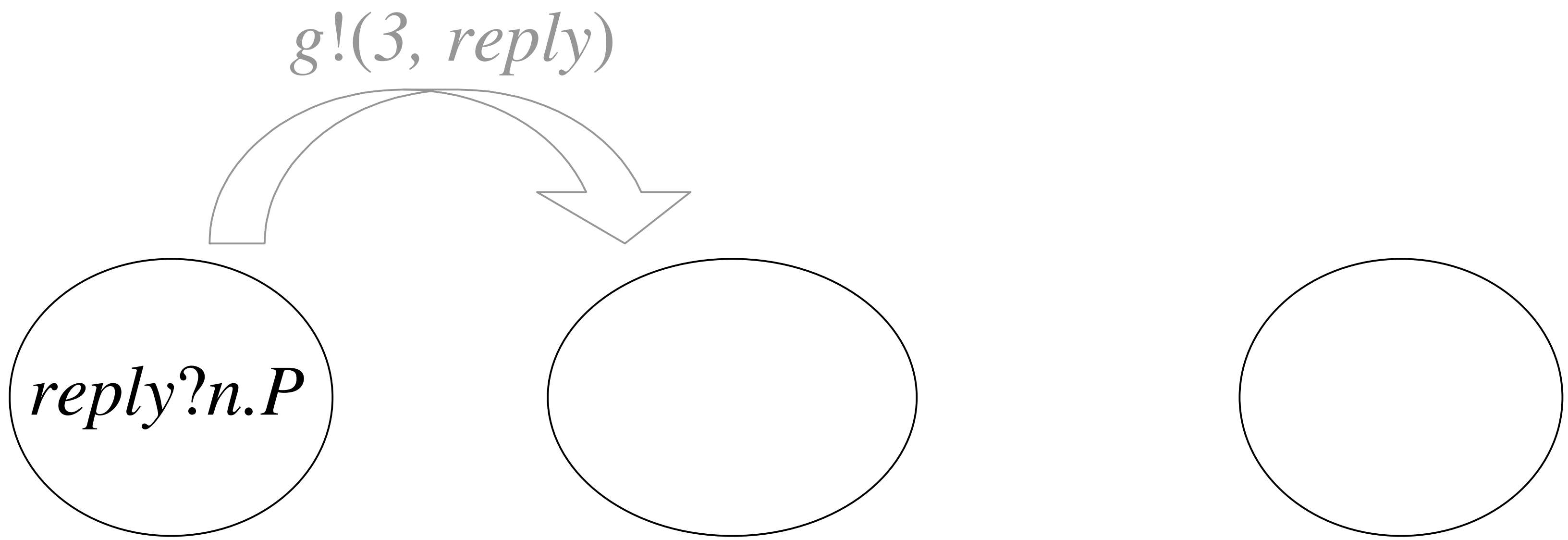
Elimination of Redundant Communication



Elimination of Redundant Communication

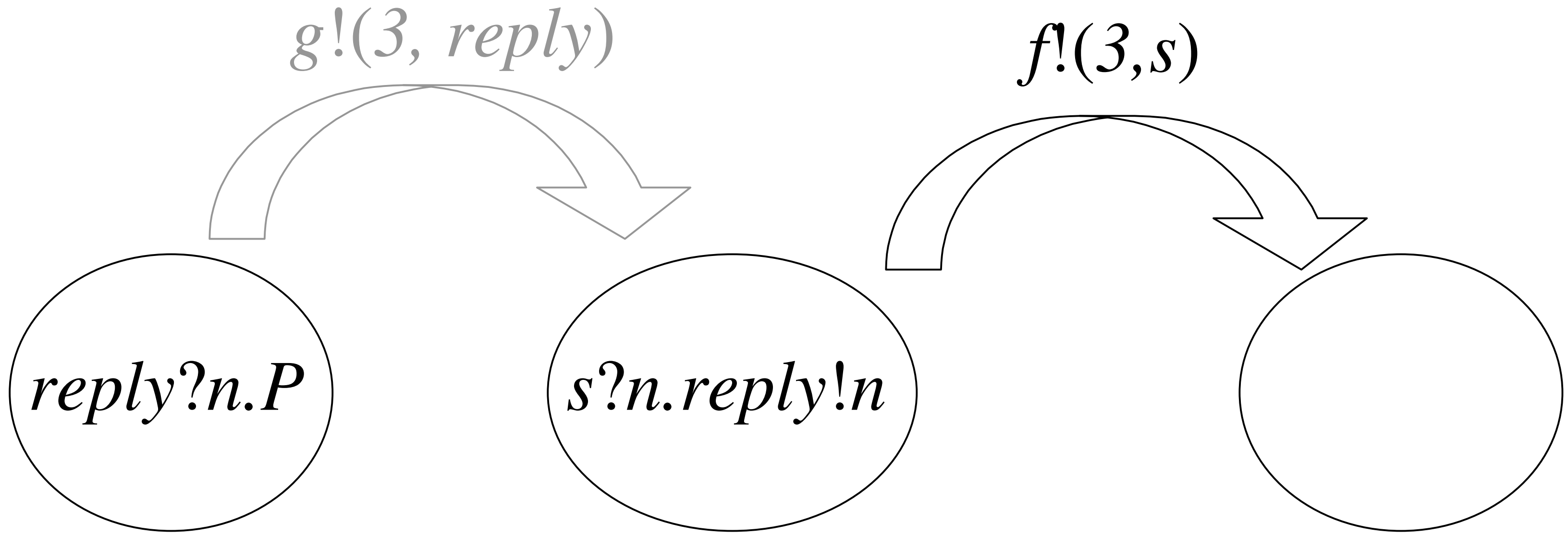


Elimination of Redundant Communication



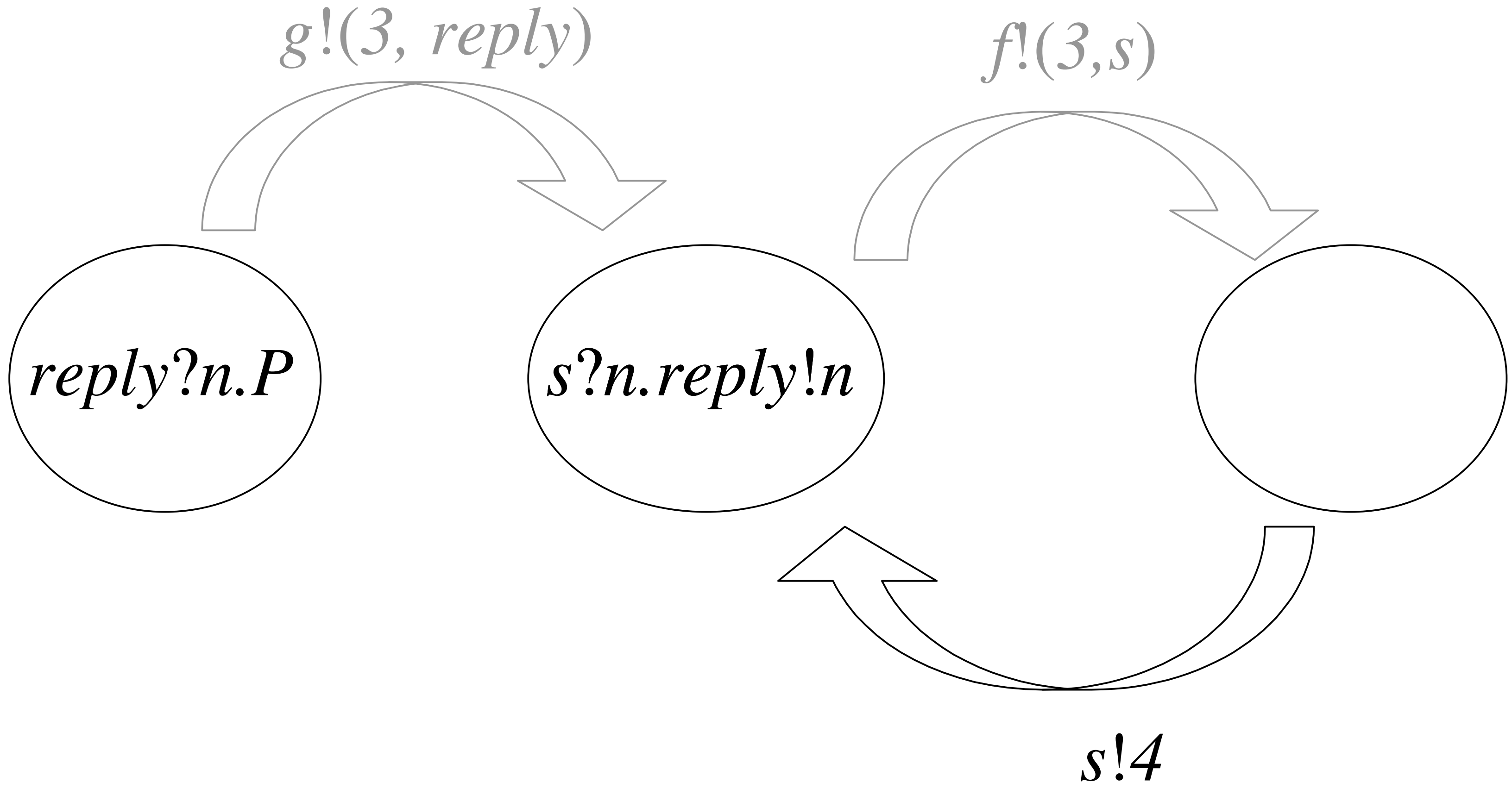
s : fresh channel

Elimination of Redundant Communication



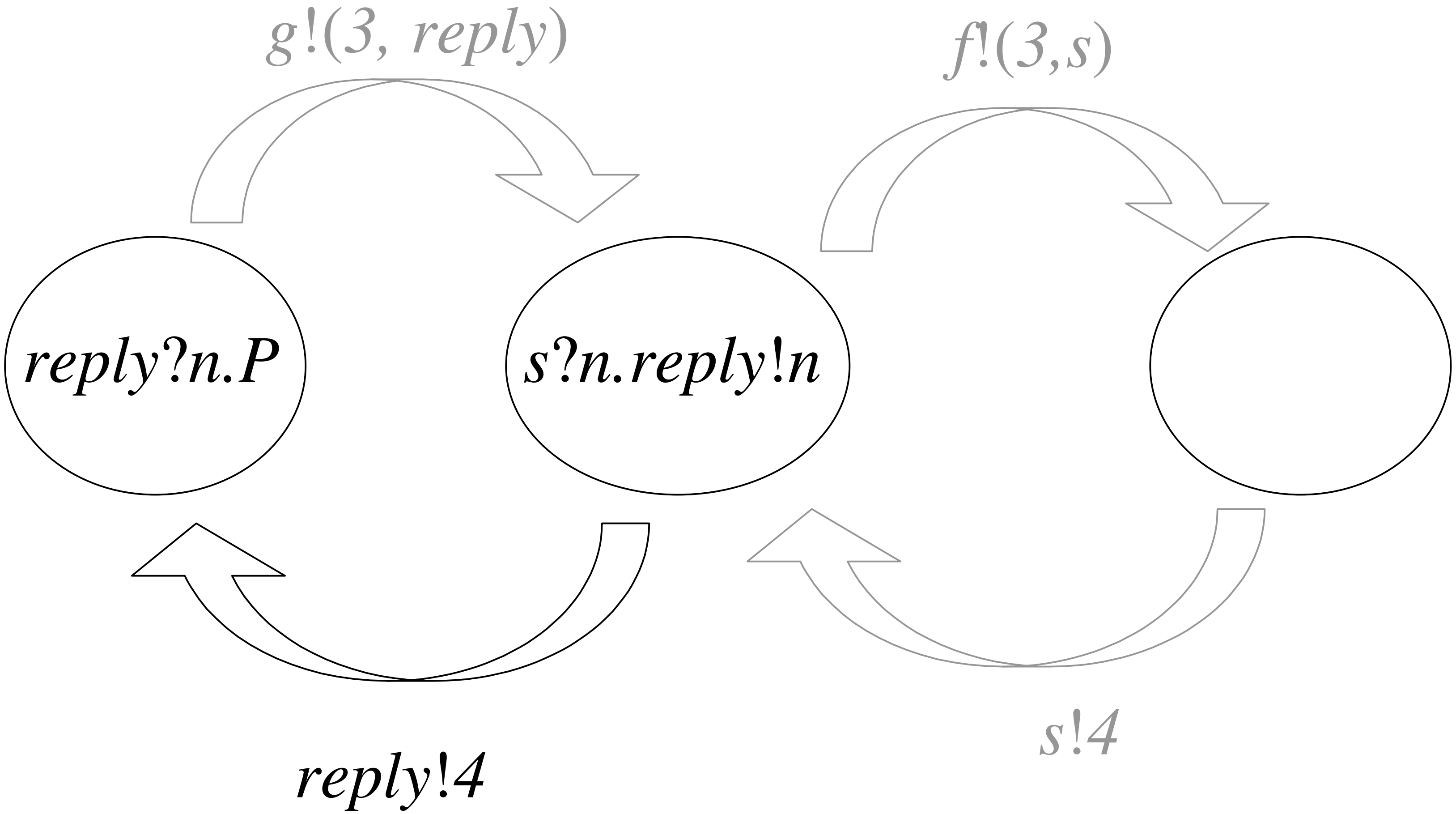
s : fresh channel

Elimination of Redundant Communication



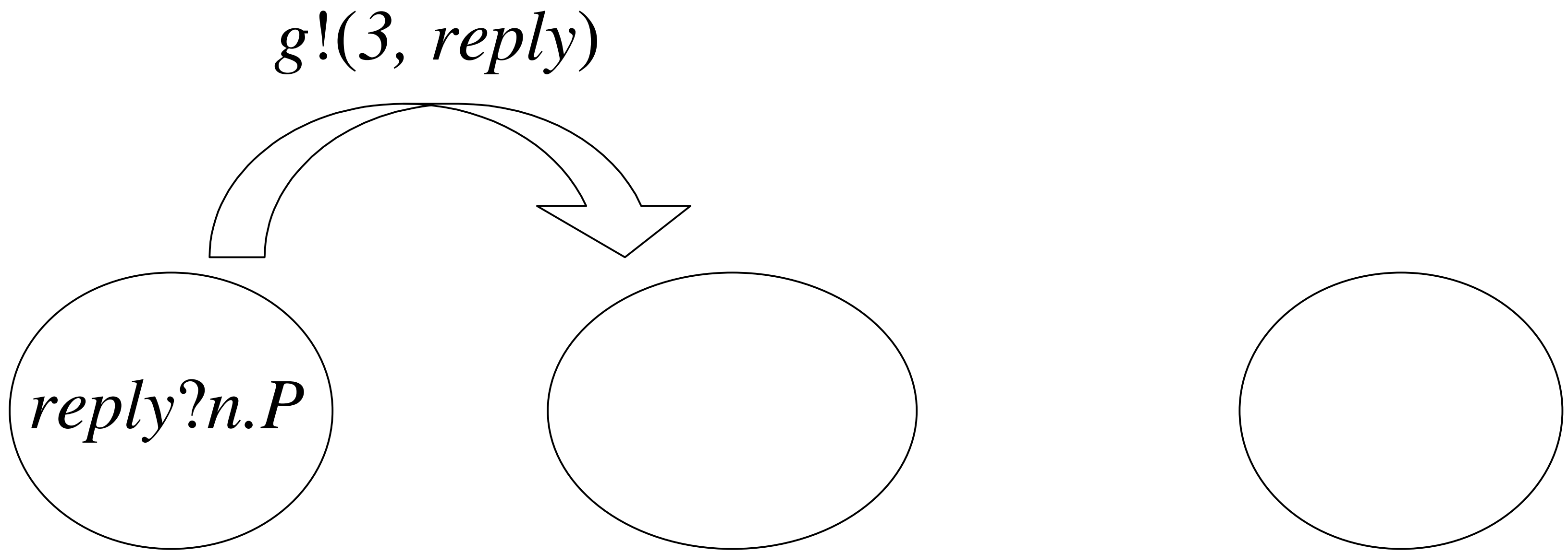
s : fresh channel

Elimination of Redundant Communication

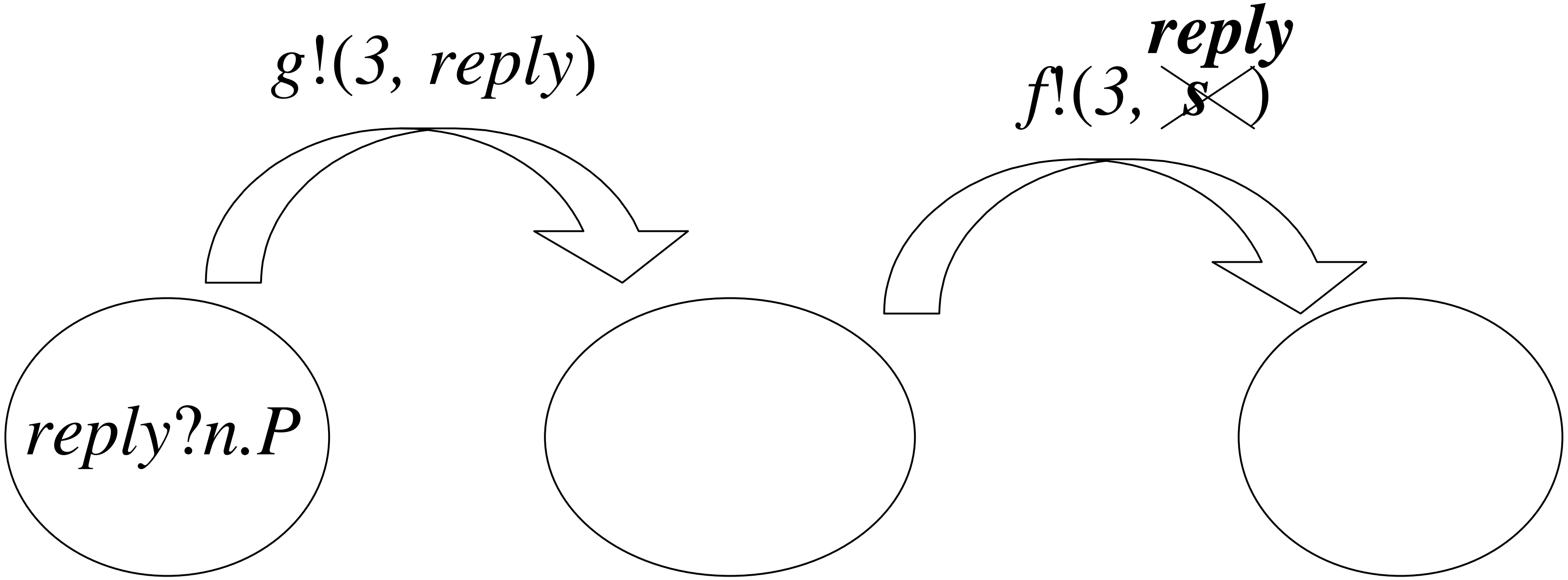


s : fresh channel

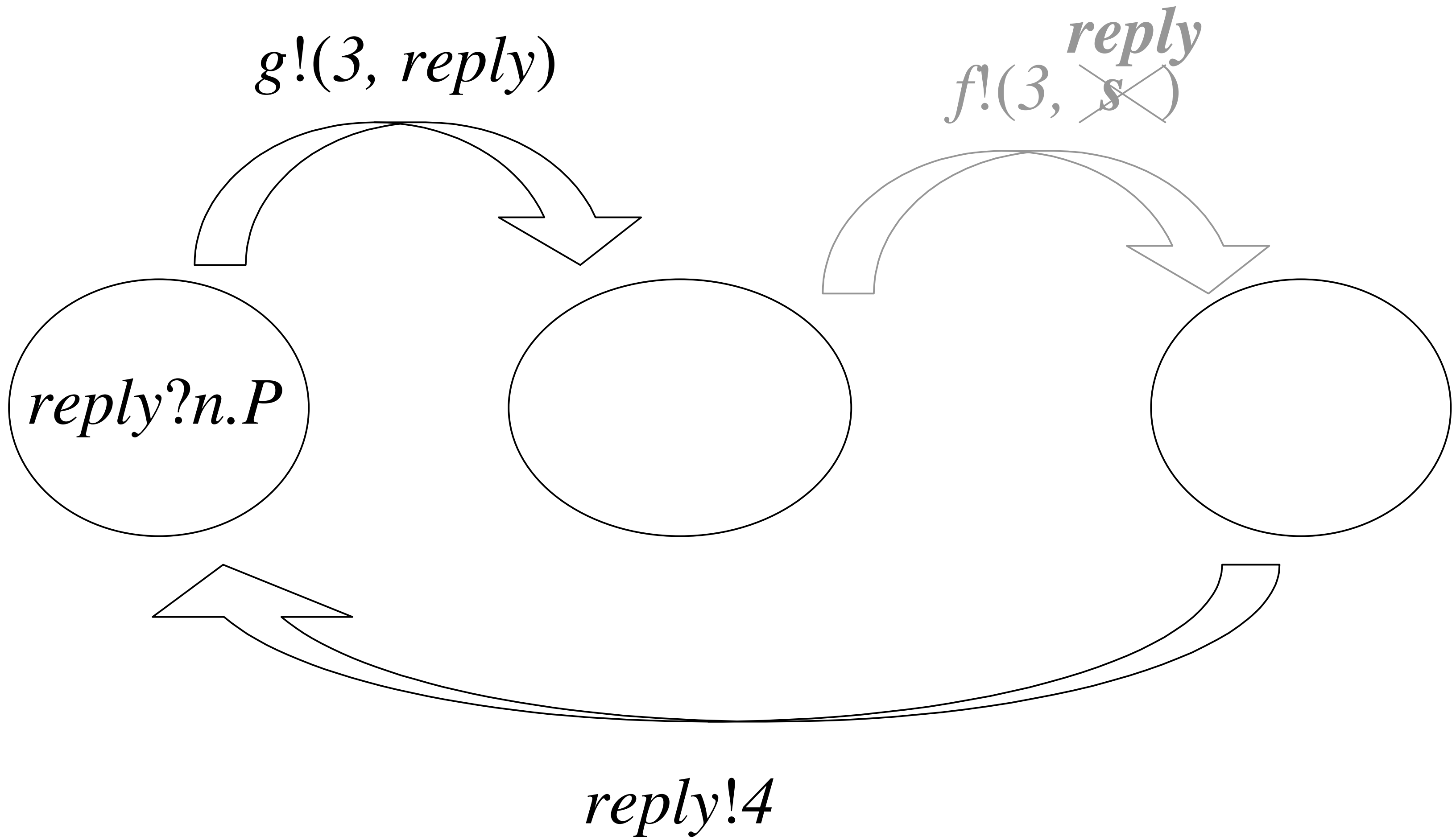
Elimination of Redundant Communication



Elimination of Redundant Communication



Elimination of Redundant Communication



Linear (Use-Once) Channel: Why Is It Useful?

- From programmers' point of view
  - Program safety: Avoid a request being ignored
  - Help for reasoning: Make the order of communications irrelevant (partial confluence)
- From compilers' point of view
  - Elimination of redundant communication

    $new\ s.(f!(v,\ s) \mid s?n.r!n) \approx f!(v,\ r)$
  - Improvement of memory usage, optimization of channel representation

Linear Channel Type System: Types and Type Judgement

- Channel types:
  - Ch(Int, <0, 1>): an integer can be sent once
  - Ch(Int, <1, ω>): an integer can be received once and sent many times
- Type judgement:

  √ x:Ch(Int, <1,0>), y:Ch(Int,<0,1>) $\vdash x?n.y!(n+1)$

  × x:Ch(Int, <0,1>), y:Ch(Int,<0,1>) $\vdash x?n.y!(n+1)$

  √ x:Ch(Ch(Int, <0, ω>), <1,0>) $\vdash x?y.(y!1 \mid y!2)$

  × x:Ch(Ch(Int, <0, 1>), <1,0>) $\vdash x?y.(y!1 \mid y!2)$

Linear Channel Type System: Type Derivation

$$\frac{x\text{: Ch(Int, <0,1>)} \vdash P \qquad x\text{:Ch(Int, <1,0>)} \vdash Q}{x\text{: Ch(Int, <0+1,1+0>)} \vdash P \mid Q}$$

Linear Channel Type System: Type Derivation

$$\frac{\Gamma \vdash P \qquad \Delta \vdash Q}{\Gamma + \Delta \vdash P \mid Q}$$

$$\frac{x\text{: Ch(Int, <1,0>)}, n\text{: Int} \vdash P}{x\text{: Ch(Int, <1+1,0>)} \vdash x?n.P}$$

Linear Channel Type System: Type Derivation

$$\frac{\Gamma \vdash P \qquad \Delta \vdash Q}{\Gamma + \Delta \vdash P \mid Q}$$

$$\frac{\Gamma, n{:}\, \tau \vdash P}{\Gamma + x{:}\ \mathrm{Ch}(\tau, \langle 1,0 \rangle) \vdash x?n.P}$$

Linear Channel Type System: Type Reconstruction

- Problem: many possible typings for a process
  - x:Ch(Ch(Int, $\langle 1,0 \rangle$),$\langle 1,0 \rangle$), y:Ch(Int,$\langle 1,1 \rangle$) $\vdash x!y \mid y!1$
  - x:Ch(Ch(Int, $\langle 0,1 \rangle$),$\langle 1,0 \rangle$), y:Ch(Int,$\langle 0,\omega \rangle$) $\vdash x!y \mid y!1$
  - x:Ch(Ch(Int, $\langle 1,1 \rangle$),$\langle 1,0 \rangle$), y:Ch(Int,$\langle 1,\omega \rangle$) $\vdash x!y \mid y!1$

    ...

$\Rightarrow$Constraint-based type system

- x:Ch(Ch(Int, $\langle i,j \rangle$),$\langle 1,0 \rangle$), y:Ch(Int,$\langle k,m \rangle$),
  $\{\boldsymbol{k >= i+1, m >= j}\} \vdash x!y \mid y!1$

Application to Compilation

Source language

new x in P

def x(y)=P in Q

...

Target language

new x in P

newL x in P

def x(y)=P in Q

defL x(y)=P in Q

...

$newL\ x\ in\ (P \mid x?z.y!z) \approx [y/x]P$

$newL\ x\ in\ (\mathrm{P} \mid x?z.Q) \approx def\ x(z) = Q\ in\ P$

...

Application to Compilation: Example

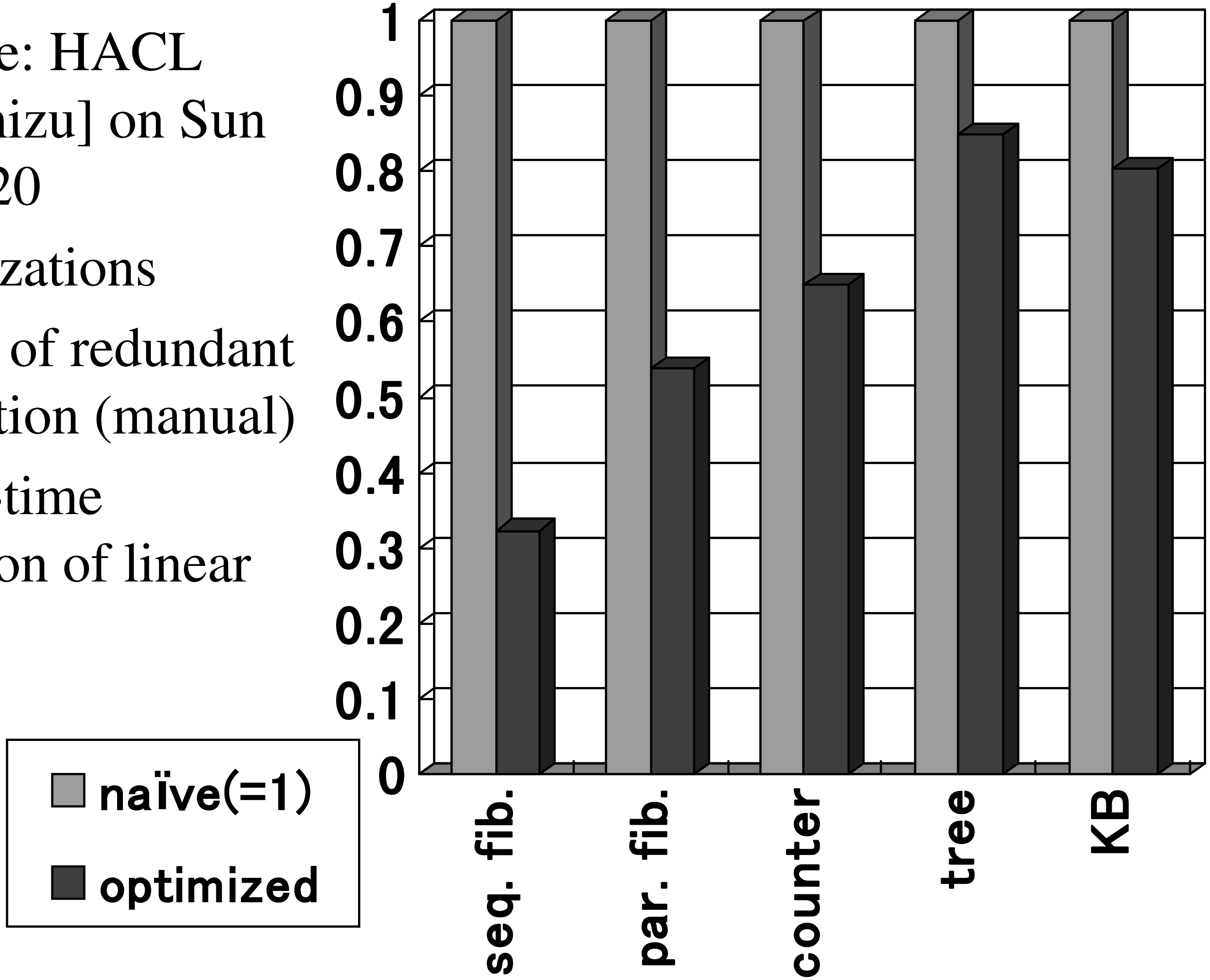
$$\mathit{def}\ \ \mathit{fib}(n,\ r) =$$
$$\mathit{new}\ \ c,\ d\ \mathit{in}\ (\mathit{fib}!(n\text{-}1,\ c) \mid \mathit{fib}!(n\text{-}2,\ d) \mid c?x.d?y.r!(x+y))$$

$$\mathit{def}\ \ \mathit{fib}(n,\ r) =$$
$$\mathit{newL}\ \ c,\ d\ \mathit{in}\ (\mathit{fib}!(n\text{-}1,\ c) \mid \mathit{fib}!(n\text{-}2,\ d) \mid c?x.d?y.r!(x+y))$$

$$\mathit{def}\ \ \mathit{fib}(n,\ r) =$$
$$\mathit{newL}\ \ d\ \mathit{in}\ \mathit{defL}\ \ c(x) = d?y.r!(x+y)\ \mathit{in}$$
$$(\mathit{fib}!(n\text{-}1,\ c) \mid \mathit{fib}!(n\text{-}2,\ d))$$

Preliminary Experiments

- Target language: HACL compiler [Shimizu] on Sun SPARCstation20
- Applied optimizations
  - Elimination of redundant communication (manual)
  - Special run-time representation of linear channels



Outline:

- ♦ Overview
- ♦ Linear Channel Type System
- ♦ ***Deadlock-Free Type System*** [Kobayashi, TOPLAS]
  - – What for?
  - – Types & type derivation
  - – Correctness -- What is guaranteed?
  - – Application to compilation

Type System for Deadlock-Freedom: Control of the Order of Communications

- Why Is It Useful?
  - From programmers' viewpoint:
    - Help for debugging (locate the source of deadlock)
    - Help for reasoning about program behavior

      E.g. x: Ch(Int, <0,1>) $\vdash P$

      $\Rightarrow$ An integer will eventually be sent to x
  - From compilers' viewpoint:
    - Optimal scheduling of processes
    - Elimination of channel status check

      E.g. $P \mid x?n.Q$ where x:Ch(Int, <0,1>) $\vdash P$

      $\Rightarrow$When P is suspended, an integer has been put in x

Deadlock-Free Type System: Types and Type Judgement

- Channel types:
  - Ch(Int, t): t describes ***when*** channels can be used.
- Type judgement:

√ x:Ch(Int, t), y:Ch(Int, s), $\boldsymbol{t<s} \vdash x?n.y!(n+1)$

× x:Ch(Int, t), y:Ch(Int, s), $\boldsymbol{t<s} \vdash y?n.x!(n-1)$

√ x:Ch(Int, t), y:Ch(Int, s), $\boldsymbol{s<t} \vdash y?n.x!(n-1)$

√ x:Ch(Int, t), y:Ch(Int, s), $\boldsymbol{s<t, t<s}$

$$\vdash y?n.x!(n-1) \mid x?n.y!(n+1)$$

√ x:Ch(Ch(Int, t)×Ch(Int, s), u), $\boldsymbol{t<s}$

$$\vdash x?(y, z).y?n.z!(n+1)$$

Deadlock-Free Type System: Type Derivation

$$\frac{\Gamma, R \vdash P \quad \Delta, R \vdash Q}{\Gamma{+}\Delta, R \vdash P \mid Q}$$

$$\frac{\begin{array}{c} w\ R\ s \\ t\ R\ s \Rightarrow u\ R\ s,\ s\ R\ t \Rightarrow s\ R\ u \\ x{:}\mathrm{Ch}(\mathrm{Int}, t), y{:}\mathrm{Ch}(\mathrm{Int}, s), R \vdash P \end{array}}{z{:}\mathrm{Ch}(\mathrm{Ch}(\mathrm{Int}, u), w), y{:}\mathrm{Ch}(\mathrm{Int}, s), R \vdash z?x.P}$$

What Is Guaranteed by Deadlock-Free Type System?

- If $\Gamma, R \vdash P$ and $P \rightarrow^* Q \not\rightarrow$, then
  - all computation has been completed, or
  - some subprocess of Q is waiting on an unreliable channel, or
  - some subprocess of Q is trying to communicate with an external process
- Deadlock-freedom is guaranteed for
  - call-by-value, call-by-name, and call-by-need λ-calculus
  - many concurrent objects

Application to Compilation

Source language		Target language
$x!v$	$\Rightarrow$	$x!v$, $x!!v$
$x?y.P$		$x?y.P$, $x??y.P$
$P \mid Q$		$P \mid Q$, $P; Q$
...		...

- $\mathit{Compile}(\Gamma\{x{:}\mathrm{Ch}(\mathrm{Int},\langle 0,1\rangle)\}, x?n.P) =$

  $$x??\, n.\mathit{Compile}(\Gamma\{n{:}\mathrm{Int}\}, P)$$

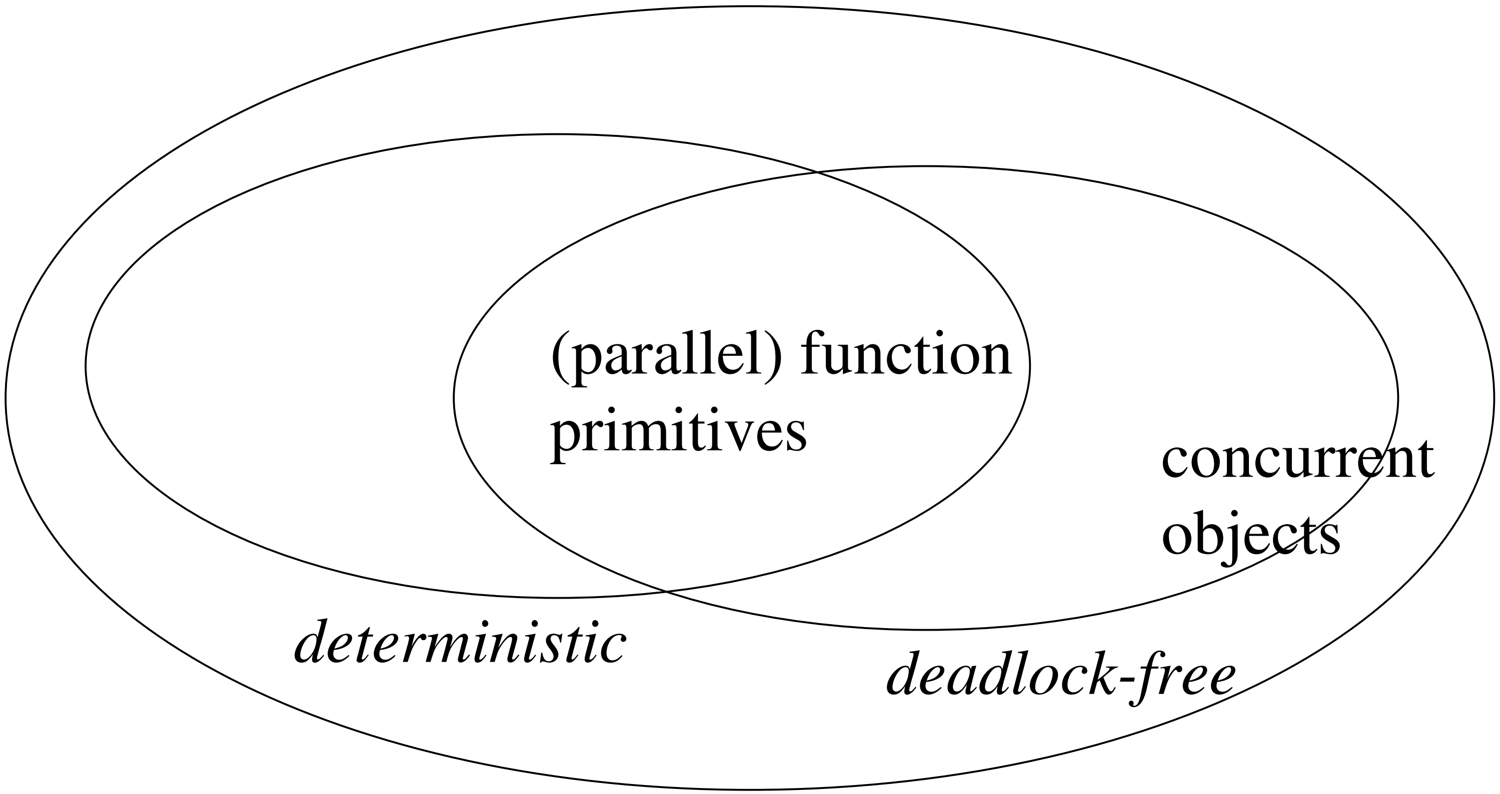
- $\mathit{Compile}(\Gamma,\ P_\Delta \mid Q_\Phi) =$

  $$\mathit{Compile}(\Gamma,\ P_\Delta);\ \mathit{Compile}(\Gamma+\Delta,\ Q_\Phi)$$

…

Conclusion

- Types are useful for inferring static information on the behavior of concurrent programs
  - *Which communication may succeed/be blocked?*
  - *Which communication introduces non-determinism?*



Conclusion

- Types are useful for inferring static information on the behavior of concurrent programs
  - *Which communication may succeed/be blocked?*
  - *Which communication introduces non-determinism?*

- Recovered information can be used for various optimizations
  - *Elimination of redundant communication*
  - *Optimal scheduling of processes*
  - *Elimination of run-time check*

Future Work

- Applications of the Type-based Techniques to Real Concurrent Programming Languages
- Redesign of Concurrent Programming Languages
  - *What should be primitives and what may be derived forms?*

    Should functions or concurrent objects be really primitives? (c.f. Pict[Pierce &Turner])
- Development of Other Type-Based Techniques
  - *Memory management* (c.f. region inference)