Processes as Types: A Generic Framework of Behavioral Type Systems for Concurrent Processes

Atsushi Igarashi (Kyoto Univ.)

based on joint work [POPL2001, TCS2003]
with Naoki Kobayashi (Tohoku Univ.)

Programming is hard ...

Concurrent programming is much harder, because...

Additional Complexity in Concurrent Programs

- Multiple threads of control
- Non-determinism
- Deadlock / livelock

Static Checking to Rescue?

Two popular approaches:

- Type Systems
  - (Said to be) good at finding 'shallow' bugs
    - e.g., arity mismatch in communication
  - Directly deal with program code
- Model Checking
  - Good at verifying 'deep' properties
    - e.g., deadlock freedom
  - Daunting(?) model extraction from programs

Previous Type Systems

- I/O mode ([Pierce&Sangiorgi 93])
  - Channels are used for correct I/O modes.
- Linearity, race conditions, atomicity ([Kobayashi, Pierce & Turner 96] [Abadi,Flanagan&Freund 99, 2000] etc.)
  - Certain communications do not suffer from non-determinism.
- Deadlock/Livelock-freedom ([Yoshida 96; Kobayashi et al.97,98,2000; Puntigam 98] etc.)
  - Certain communications succeed eventually.
- Security properties ([Honda, Vasconcelos & Yoshida 2000; Hennessy & Riely 2000; Kobayashi 2005] etc.)

Problems of Previous Type Systems for Concurrent Programs

- Designed in an ad hoc manner
  - Unclear essence
  - Difficulty of integrating different type systems.
  - A lot of repeated work:
    - type soundness proofs
    - type inference algorithms

⇐ No common framework

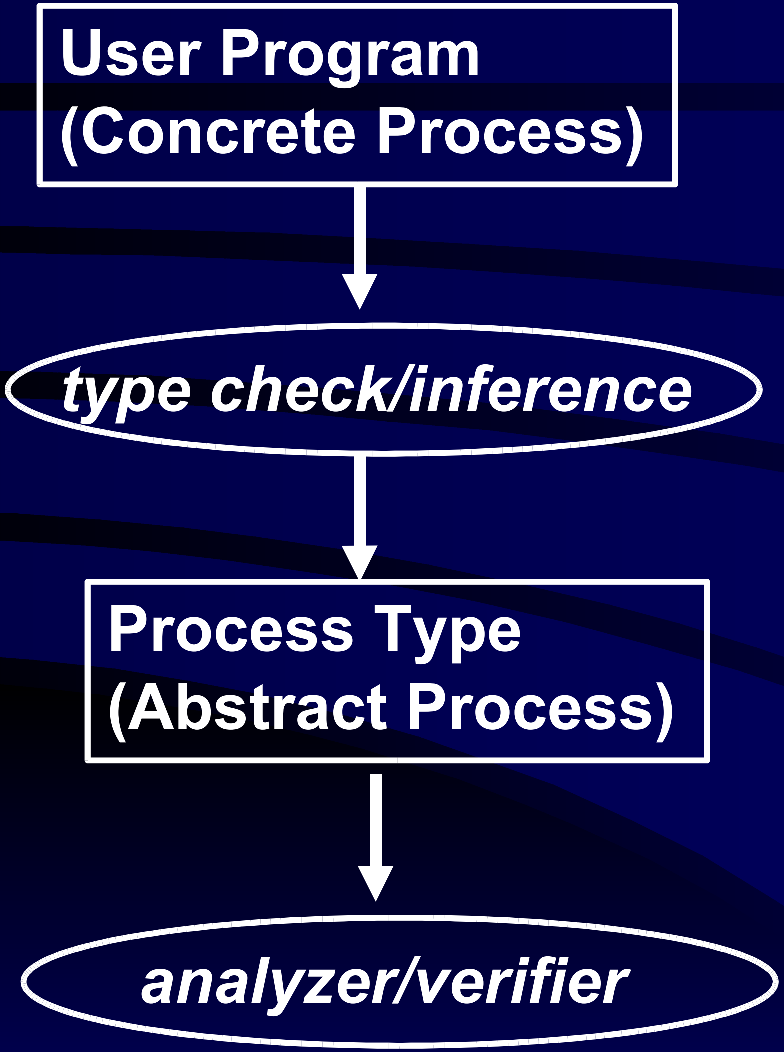
c.f. Curry-Howard isomorphism, Effect systems

This Talk:

Generic Type System

- Provides a common framework of type systems for concurrent programs
- Can be instantiated easily to various type systems (e.g., for race-freedom, deadlock-freedom)
- Enables sharing of a large amount of work for development of type systems
  - type soundness proofs
  - type inference algorithms

Idea: Types as Abstract Processes



User Program
(Concrete Process)

type check/inference

Process Type
(Abstract Process)

analyzer/verifier

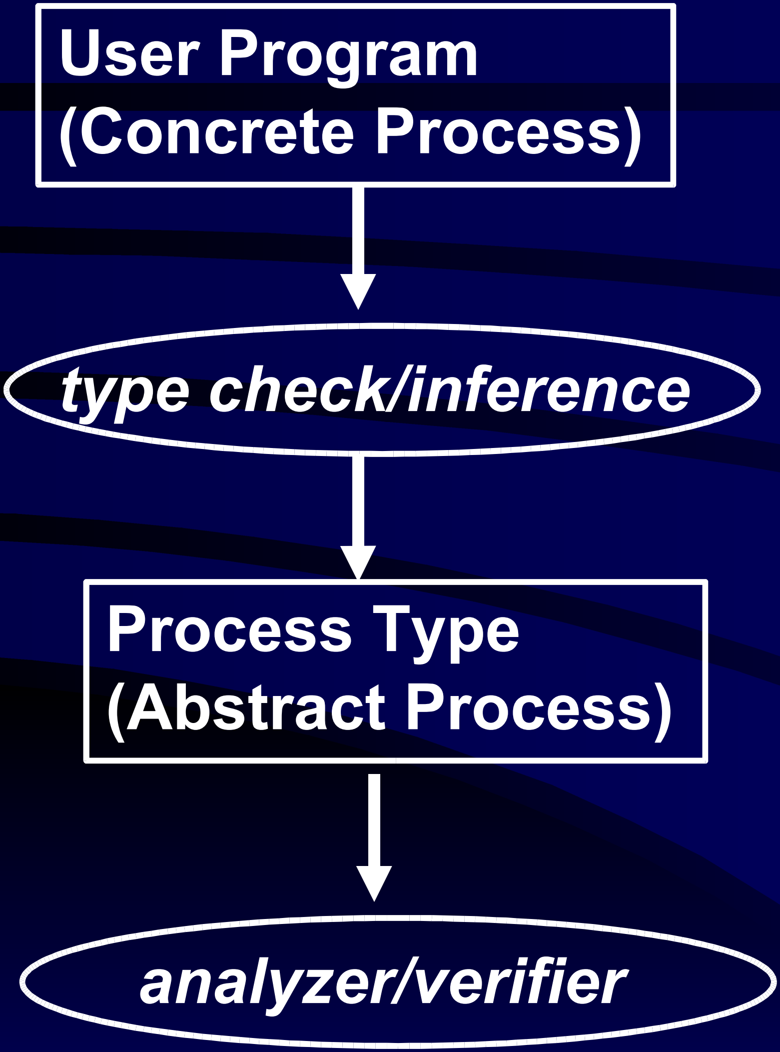
race

$$\begin{tikzcd}
P_1 \arrow[r] \arrow[d, leftrightarrow] & P_2 \arrow[r] \arrow[d, leftrightarrow] & \ldots \arrow[r] & P_n \arrow[d, leftrightarrow] \\
\Gamma_1 \arrow[r] & \Gamma_2 \arrow[r] & \ldots \arrow[r] & \Gamma_n
\end{tikzcd}$$

race

c.f. Abstract Interpretation [Cousot&Cousot77]

Idea: Types as Abstract Processes



User Program (Concrete Process)

type check/inference

Process Type (Abstract Process)

analyzer/verifier

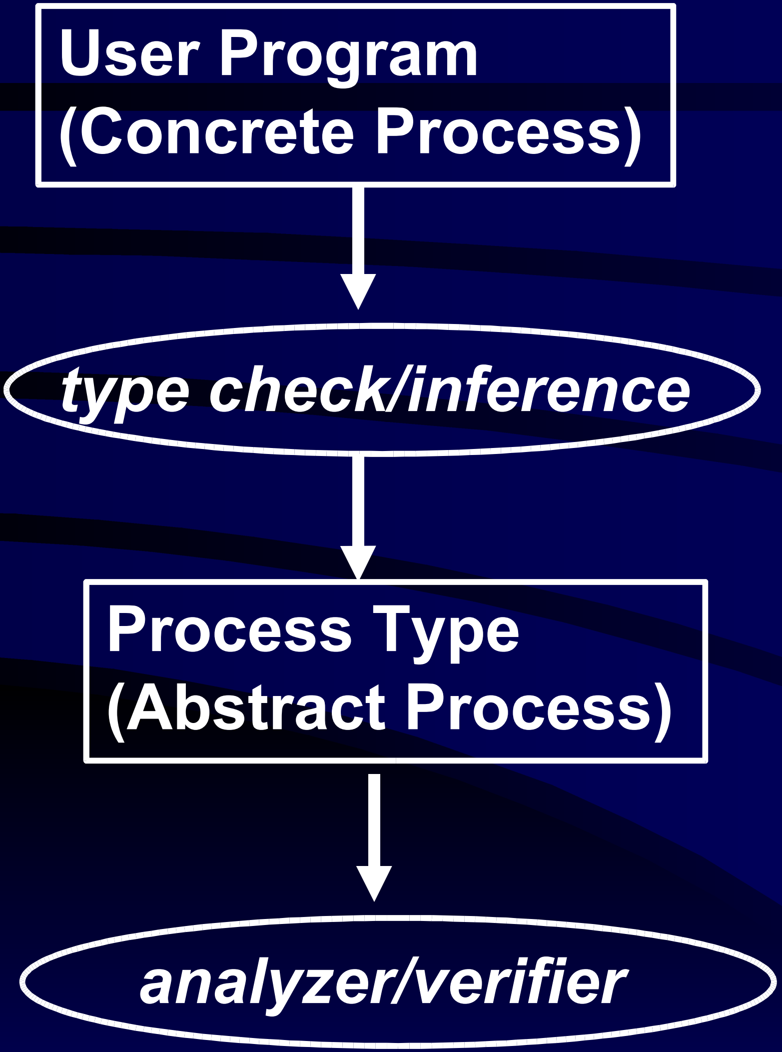
deadlock

$$\begin{tikzcd}
P_1 \arrow[r] \arrow[d, leftrightarrow] & P_2 \arrow[r] \arrow[d, leftrightarrow] & \ldots \arrow[r] & P_n \arrow[d, leftrightarrow] \\
\Gamma_1 \arrow[r] & \Gamma_2 \arrow[r] & \ldots \arrow[r] & \Gamma_n
\end{tikzcd}$$

deadlock

c.f. Abstract Interpretation [Cousot&Cousot77]

Idea: Types as Abstract Processes



User Program
(Concrete Process)

type check/inference

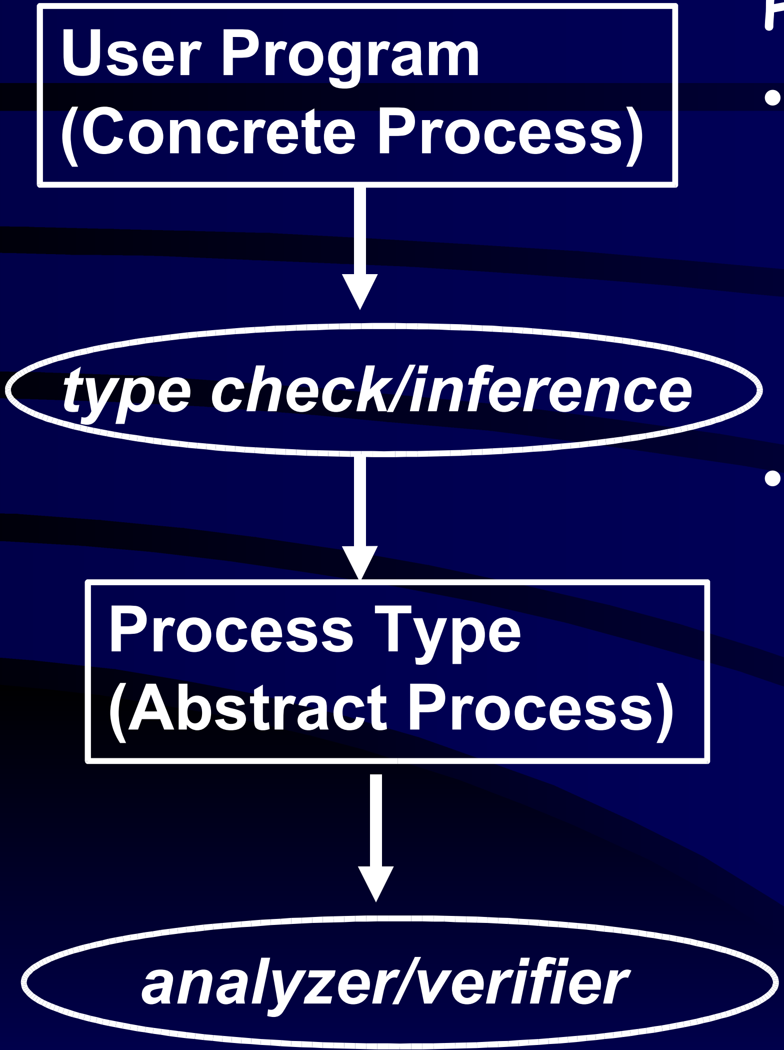
Process Type
(Abstract Process)

analyzer/verifier

π-calculus[Milner et al.]:
Dynamic change of
communication toplology
⇒Expressive,
but hard to analyze

CCS (w/o channel creation)
No dynamic change of
communication topology
⇒Much easier to analyze

Idea: Types as Abstract Processes



User Program
(Concrete Process)

type check/inference

Process Type
(Abstract Process)

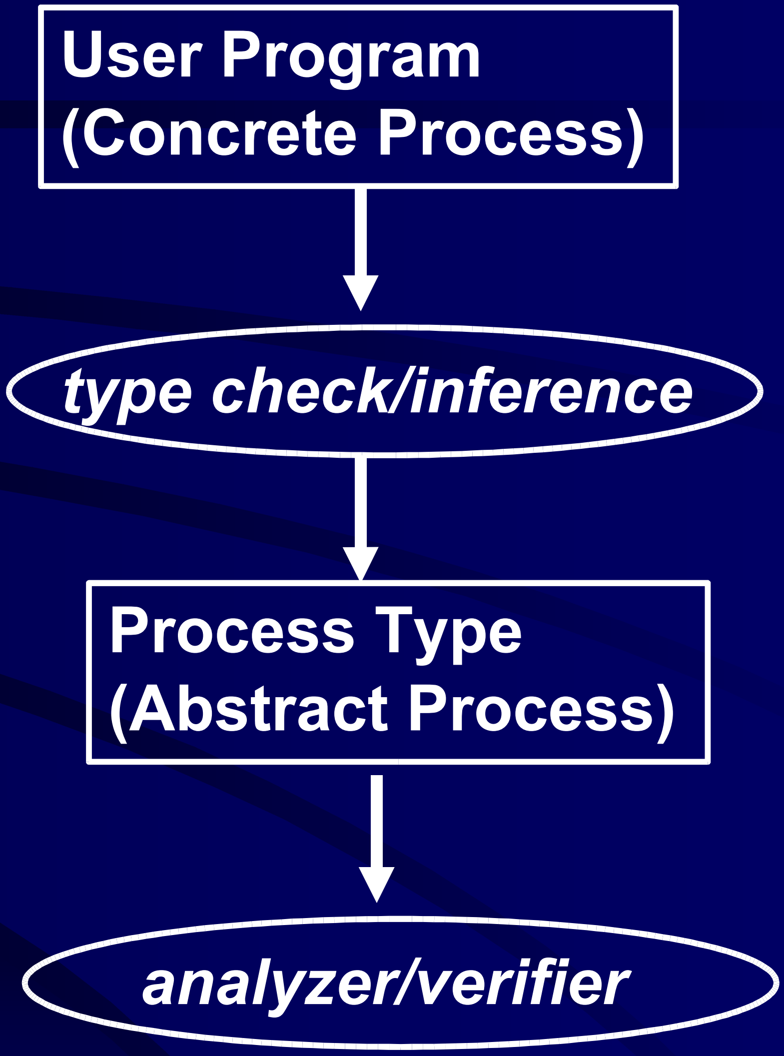
analyzer/verifier

Hybrid approach combining

- Type systems
  - Type inference as syntax-directed, automatic model extraction from a program
- Model checking
  - Analyzer/verifier as model checker

Outline

- Target Language
  - Syntax
  - Operational semantics
- Process Types
- Generic Type System
  - Typing rules
  - Type soundness
- How to obtain specific type systems



User Program (Concrete Process)

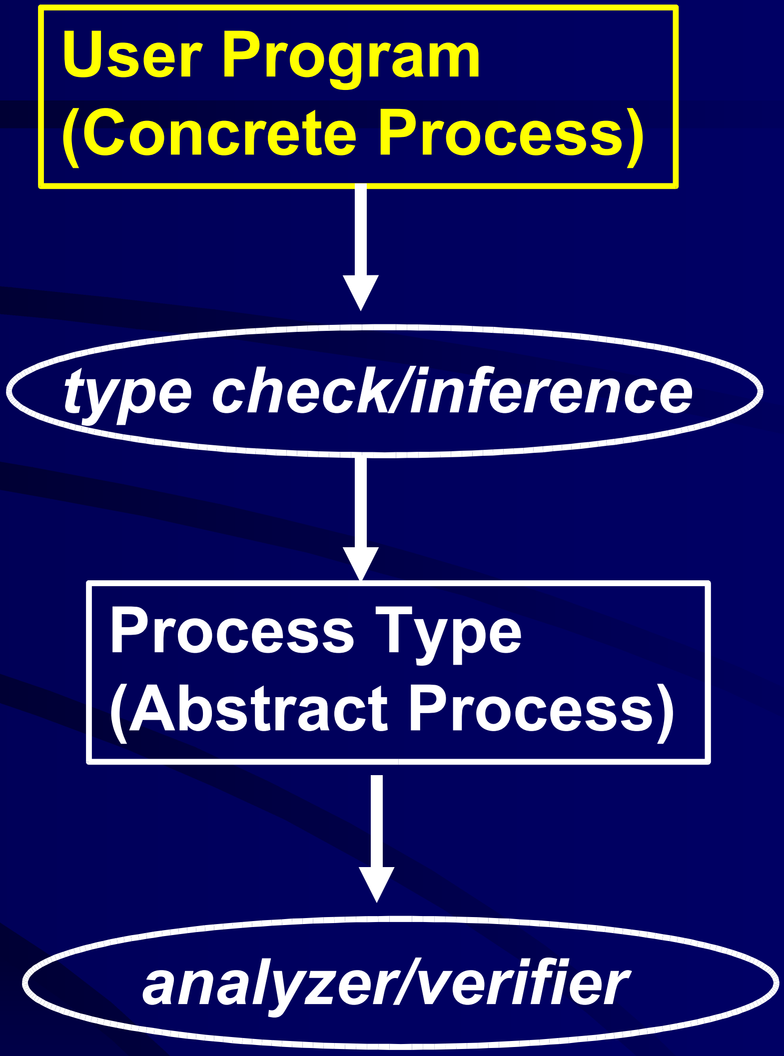
type check/inference

Process Type (Abstract Process)

analyzer/verifier

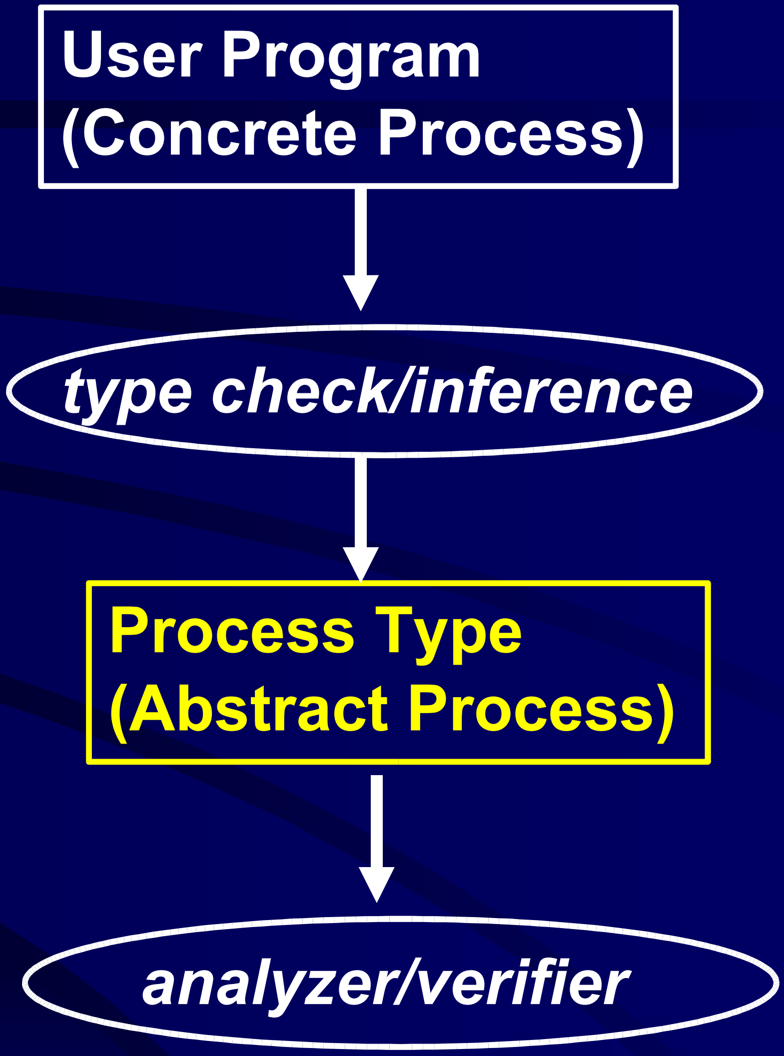
Outline

- Target Language
  - Syntax
  - Operational semantics
- Process Types
- Generic Type System
  - Typing rules
  - Type soundness
- How to obtain specific type systems



Outline

- Target Language
  - Syntax
  - Operational semantics
- Process Types
- Generic Type System
  - Typing rules
  - Type soundness
- How to obtain specific type systems



User Program
(Concrete Process)

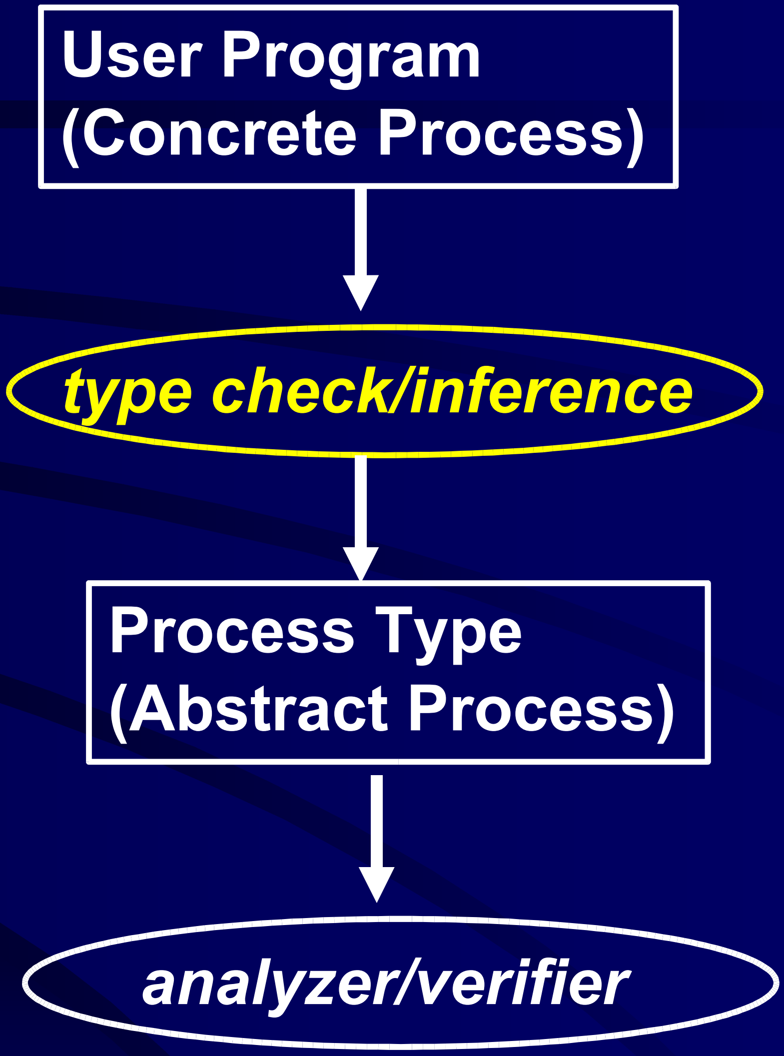
type check/inference

Process Type
(Abstract Process)

analyzer/verifier

Outline

- Target Language
  - Syntax
  - Operational semantics
- Process Types
- Generic Type System
  - Typing rules
  - Type soundness
- How to obtain specific type systems



User Program (Concrete Process)

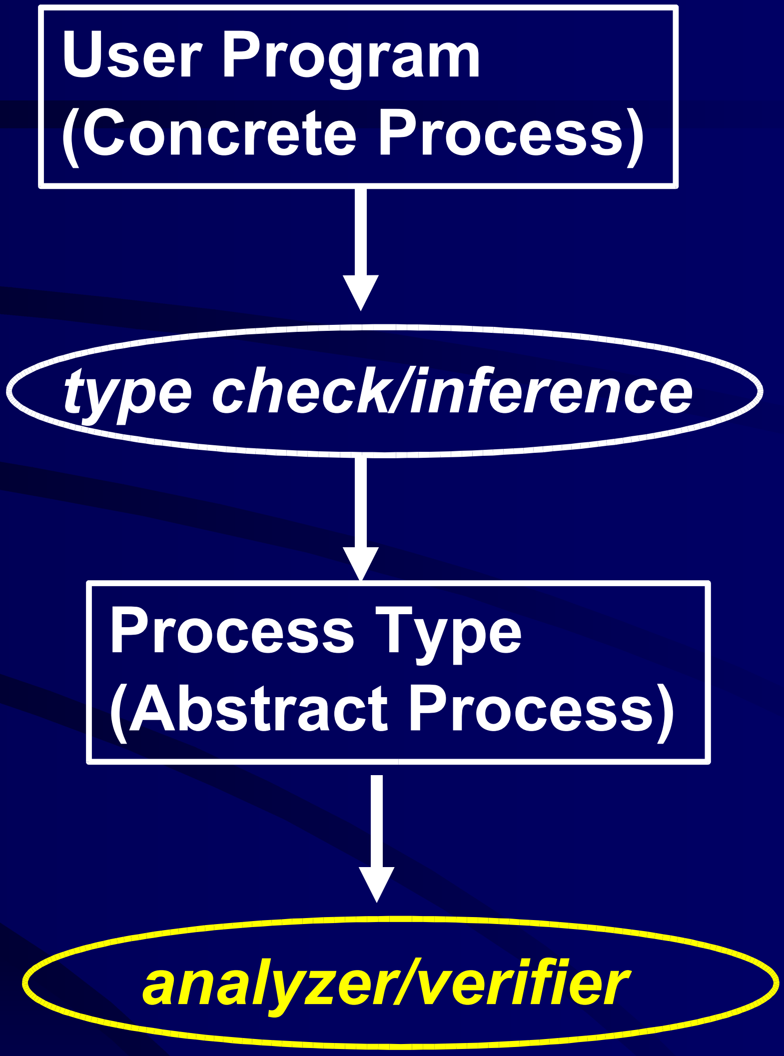
type check/inference

Process Type (Abstract Process)

analyzer/verifier

Outline

- Target Language
  - Syntax
  - Operational semantics
- Process Types
- Generic Type System
  - Typing rules
  - Type soundness
- How to obtain specific type systems



User Program
(Concrete Process)

type check/inference

Process Type
(Abstract Process)

analyzer/verifier

Target Language: π-calculus[Milner et al.]

Calculus of concurrent processes with:

- Message passing via communication channels
- First-class channels
- Dynamically created channels
- Infinite behavior by replication

Target Language: π-calculus[Milner et al.]

P, Q (Processes) ::=

0	(inaction)
$x!^{t}[\ v_1, \ldots, v_n\ ].\ P$	(output)
$x?^{t}[\ y_1, \ldots, y_n\ ].\ P$	(input)
new $x_1, \ldots, x_n$ in P	(channel creation)
$P \mid Q$	(parallel execution)
$*P$	(replication: $\approx$ P | P | ...)

s, t : labels to identify program points

$$x![v_1,\ldots,v_n].P \mid x?[y_1,\ldots,y_n].Q \rightarrow P \mid [v_1/y_1,\ldots,v_n/y_n]Q$$

(c.f. β-reduction: $(\lambda x.M)N \rightarrow [N/x]M$)

Example: Function Server

Server: $*\mathit{succ}?[n, r].r\,![n+1]$

Client: new r in $(\mathit{succ}![1,r] \mid r?\,[x]\ldots)$

Example: Function Server

Server: $*\mathit{succ}?[n, r].r\,![n+1]$

Client: $\text{new } r' \text{ in } (\mathit{succ}![1,r] \mid r?\,[x]\ldots)$

$$\underbrace{*\mathit{succ}?[n, r].r![n+1]}_{\text{server}} \mid \underbrace{\text{new } r' \text{ in } (\mathit{succ}![1,r'\,] \mid r'\,?[x].\mathit{print}![x])}_{\text{client}}$$

Example: Function Server

Server: $*succ?[n, r].r\,![n+1]$

Client: new r in $(succ![1,r] \mid r?\,[x]\ldots)$

$$\underbrace{*succ?[n, r].r![n+1]}_{\text{server}} \mid \underbrace{\text{new } r' \text{ in } (succ![1,r'] \mid r'?[x].print![x])}_{\text{client}}$$

$$\rightarrow *succ?[n, r].r![n+1] \mid \text{new } r' \text{ in } (r'![2] \mid r'?[x].print![x])$$

Example: Function Server

Server: $\ast succ?[n, r].r\,![n+1]$

Client: $\text{new } r \text{ in } (succ![1,r] \mid r?\,[x]\ldots)$

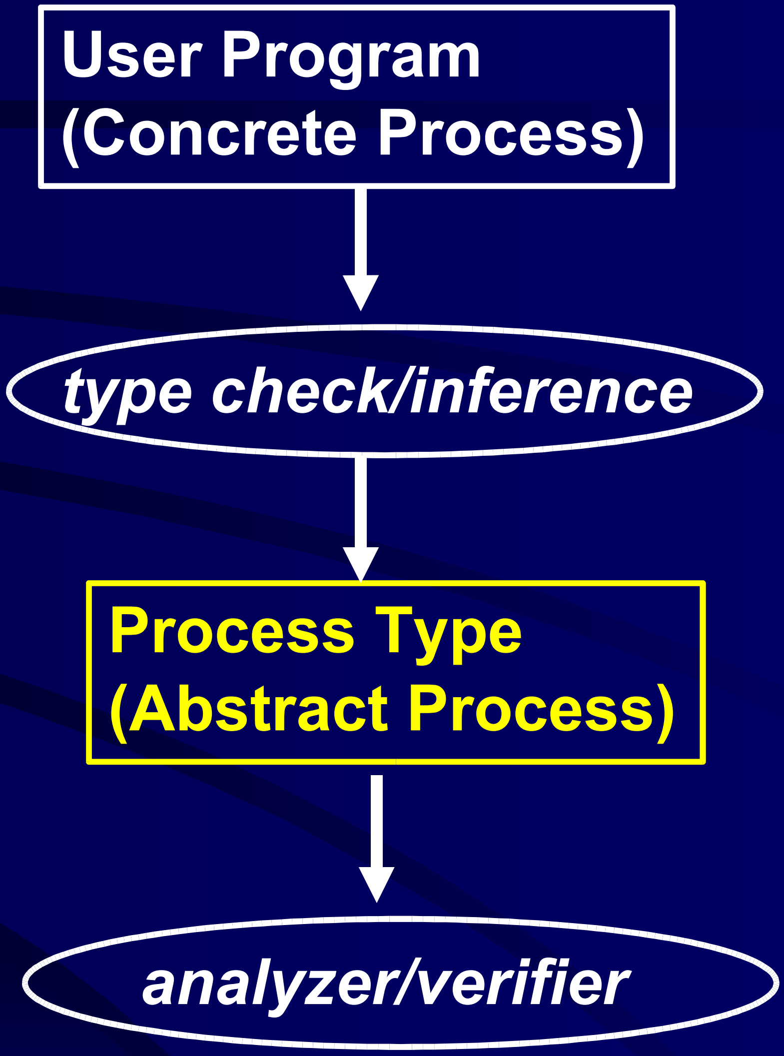
$$\underbrace{\ast succ?[n, r].r![n+1]}_{\text{server}} \mid \underbrace{\text{new } r' \text{ in } (succ![1,r'] \mid r'?[x].print![x])}_{\text{client}}$$

$$\rightarrow \ast succ?[n, r].r![n+1] \mid \text{new } r' \text{ in } (r'\,![2] \mid r'\,?[x].print![x])$$

$$\rightarrow \ast succ?[n, r].r![n+1] \mid print![2]$$

Outline

- Target Language
  - Syntax
  - Operational semantics
- Process Types
- Type System
  - Typing rules
  - Type soundness
- Instances of Generic Type System



User Program (Concrete Process)

type check/inference

Process Type (Abstract Process)

analyzer/verifier

Process Types

Γ, Δ (process types) ::= 0 (inaction)

- $x!^{t}[\tau].\ \Gamma$ (output a value on x, then behaves like Γ)
- $x?^{t}[\tau].\ \Gamma$ (input from x, then behaves like Γ)
- $t.\Gamma$ (wait for event t, then behaves like Γ)
- $\Gamma|\Delta$ (parallel composition)
- $*\Gamma$ (replication)
- $\Gamma \,\&\, \Delta$ (non-deterministic choice)

τ (tuple types) ::=

- $(x_1,\ldots,x_n)\Gamma$ (type of a tuple of the form $[x_1,\ldots,x_n]$, which should be used according to Γ)

Examples

- $x?^{s}[\mathrm{Int}].y!^{t}[\mathrm{Int}]$
  - Receives an integer through x and then sends an integer through y

    (e.g. $x?^{s}[n].y!^{t}[n+1]$)
- $*x?^{s}[(y)y!^{t}[\mathrm{Int}]]$
  - Repeatedly receives a channel and sends an integer through the received channel

    (e.g. $*x?^{s}[y].y!^{t}[2]$)

Examples

- $\mu\alpha.$***put*****?[Int].*****get*****?[(*****y*****)** ***y*****![Int]].**α
  - The type of a one-place buffer:
    - ***Buffer*** **=** ***put*****?[*****n*****].*****get*****?[*****r*****].*****r*****![*****n*****].*****Buffer***
      - (expressed by using replication)
    - $\mu\alpha.\Gamma$ ... recursive process type that satisfies $\mu\alpha.\Gamma = [\mu\alpha.\Gamma/\alpha]\Gamma$
      - ($*\Gamma$ is actually a syntax sugar using μ)

Process Types Form a Mini-Process Calculus

$x![\tau].\Gamma \mid x?[\tau].\Delta \;\rightarrow\; \Gamma \mid \Delta$

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

……

e.g. $x![\tau].y![\mathrm{Int}] \mid x?[\tau] \rightarrow y![\mathrm{Int}]$

($\tau = (z)z![\mathrm{Int}]$)

$x![y] \mid x?[z].z![2] \;\rightarrow\; y![2]$

Summary of Processes and Process Types

$x!^t[v_1,\ldots,v_n].P$	output	$x!^t[\tau].\Gamma$
$x?^t[y_1,\ldots,y_n].P$	input	$x?^t[\tau].\Gamma$
$P\|Q$	parallel	$\Gamma\|\Delta$
new $x_1,\ldots,x_n$ in P	new channel	
$*P$	replication	$*\Gamma$
	wait	$t.\Gamma$
	non-determinism	$\Gamma \& \Delta$

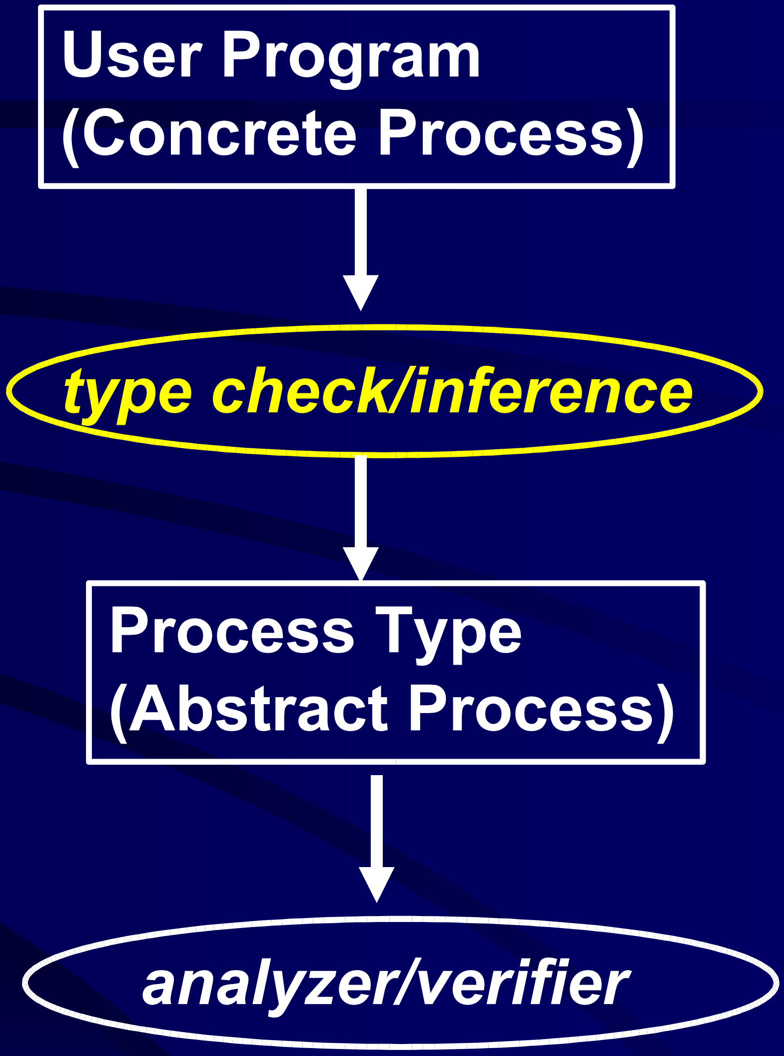
Summary of Proces

No value passing,
only synchronization behavior

$x!^t[v_1,\ldots,v_n].P$	output	$x!^t[\tau].\Gamma$
$x?^t[y_1,\ldots,y_n].P$	input	$x?^t[\tau].\Gamma$
$P \mid Q$	parallel	$\Gamma \mid \Delta$
new $x_1,\ldots,x_n$ in P	new channel	
$*P$	replication	$*\Gamma$
	wait	$t.\Gamma$
	non-determinism	$\Gamma\&\Delta$

Outline

- Target Language
  - Syntax
  - Operational semantics
- Process Types
- Type System
  - Typing rules
  - Type soundness
- Instances of Generic Type System



User Program (Concrete Process)

type check/inference

Process Type (Abstract Process)

analyzer/verifier

Recipe for Type Systems

- Type judgment relation
  - with supposed meaning
- Typing rules to derive type judgments
  - One rule for one syntactic construct
- Type soundness theorem
  - Evidence that the supposition is indeed true
- (Type inference algorithm)

Type Judgment

$\Gamma \vdash P$

P has process type Γ

Γ is an abstraction of P

P matches specification Γ

Examples:

- $x?^{s}[\mathrm{Int}].y!^{t}[\mathrm{Int}] \vdash x?^{s}[n].y!^{t}[n+1]$
- $*x?^{s}[(y)y!^{t}[\mathrm{Int}]] \vdash *x?^{s}[y].y!^{t}[2]$

Summary of Processes and Process Types

$x!^t[v_1,\ldots,v_n].P$	output	$x!^t[\tau].\Gamma$
$x?^t[y_1,\ldots,y_n].P$	input	$x?^t[\tau].\Gamma$
$P\|Q$	parallel	$\Gamma\|\Delta$
new $x_1,\ldots,x_n$ in P	new channel	
$*P$	replication	$*\Gamma$
	wait	$t.\Gamma$
	non-determinism	$\Gamma\&\Delta$

Typing Rule (parallel composition)

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

(similarly for $*$)

Summary of Processes and Process Types

$x!^t[v_1,\ldots,v_n].P$	output	$x!^t[\tau].\Gamma$
$x?^t[y_1,\ldots,y_n].P$	input	$x?^t[\tau].\Gamma$
$P\|Q$	parallel	$\Gamma\|\Delta$
new $x_1,\ldots,x_n$ in P	new channel	
$*P$	replication	$*\Gamma$
	wait	$t.\Gamma$
	non-determinism	$\Gamma\&\Delta$

Typing Rule (Output)

$$\frac{\Gamma \vdash P}{x!^{t}[(y)\Delta].(\Gamma \mid [v/y]\Delta) \vdash x!^{t}[v].P}$$

Δ expresses how the receiver uses y

Example:

$$x!^{t}[\tau].y!^{u}[\mathrm{Int}] \mid x?^{s}[\tau] \vdash \mathbf{x!^{t}[y]} \mid x?^{s}[z].z!^{u}[2]$$

(**for** $\tau = (z)z!^{u}[\mathrm{Int}]$)

Typing Rule (Input)

$$\frac{\Gamma \mid \Delta \vdash P \qquad y \notin FV(\Gamma)}{x!^{t}[(y)\Delta].\Gamma \vdash x?^{t}[y].P}$$

Example:

$$x!^{t}[\tau].y!^{u}[\mathrm{Int}] \mid x?^{s}[\tau] \vdash x!^{t}[y] \mid x?^{s}[z].z!^{u}[2]$$

(for $\tau = (z)z!^{u}[\mathrm{Int}]$)

Typing Rule (Channel Creation)

$$\frac{\Gamma \vdash P \qquad ok(\Gamma\downarrow\{x_1,\dots,x_n\})}{\Gamma\uparrow\{x_1,\dots,x_n\} \vdash \text{new } x_1,\dots,x_n \text{ in } P}$$

$ok(\Gamma\downarrow\{x_1,\dots,x_n\})$:

Check that $x_1,\dots,x_n$ are used 'appropriately'

(Depending on

$\Gamma\uparrow\{x_1,\dots,x_n\}$: Fo

$$(x?^t[\tau].\Gamma)\downarrow\{x\} = t.(\Gamma\downarrow\{x\})$$

Possible blocking recorded by $t.\Gamma$
(for deadlock analysis)

Typing Rule (Subsumption)

$$\frac{\Gamma \vdash P \qquad \Gamma' \leq \Gamma}{\Gamma' \vdash P}$$

Process type can be replaced with a coarser abstraction

- $\Gamma' \leq \Gamma$: Subtyping relation
  - Depends on type system instances

Weak Type Soundness Theorem (Subject Reduction)

$$\begin{tikzcd}
\Gamma \arrow[d] & \vdash & P \arrow[d] \\
\Delta & \vdash & Q
\end{tikzcd}$$

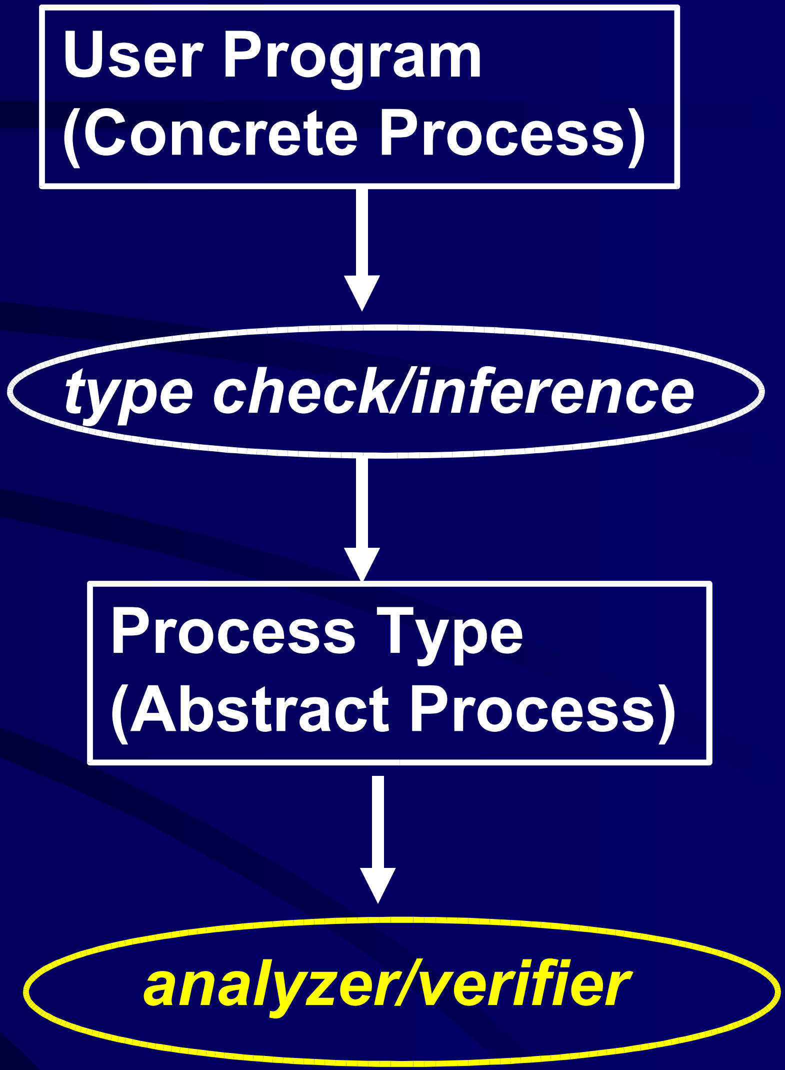
Γ simulates the behavior of P

General Type Inference

- Principal typing theorem:
  For any process P, there is a "most general" type Γ s.t. $\Gamma \vdash P$
  - (after a slight (routine) modification of the type system)

Outline

- Target Language
- Process Types
- Generic Type System
- Instances
  - Analysis of race, deadlock
  - What type system can be obtained, and how?



User Program (Concrete Process)

type check/inference

Process Type (Abstract Process)

analyzer/verifier

Type System for Race-freedom

- A process P is race-free on output if:

$$P \not\rightarrow^{*} \text{new } y_1,\ldots,y_n \text{ in } (x![\ldots].Q_1 \mid x![\ldots].Q_2 \mid R)$$

- A type Γ is race-free on output if:

$$\Gamma \not\rightarrow^{*} x!^{s}[\tau].\Delta_1 \mid x!^{t}[\tau].\Delta_2 \mid \Theta$$

Theorem: If $\Gamma \vdash P$, Γ is race-free, and *ok*(Δ) implies Δ is race-free, then P is race-free

General Principle

$$\begin{tikzcd}
P \arrow[r] \arrow[d, phantom, "\perp"] & P_1 \arrow[r] \arrow[d, phantom, "\perp"] & P_2 \arrow[r] \arrow[d, phantom, "\perp"] & \dots \arrow[r] & P_n \arrow[d, phantom, "\perp"] \\
\Gamma \arrow[r] & \Gamma_1 \arrow[r] & \Gamma_2 \arrow[r] & \dots \arrow[r] & \Gamma_n
\end{tikzcd}$$

To verify a property of P,

verify the corresponding property of Γ

Example: Race Analysis

$x![1] \mid y![x] \mid y?[z].z![2]$

$\perp$

$x![\mathrm{Int}] \mid y![\tau].\ x![\mathrm{Int}] \mid y?[\tau] \quad \rightarrow x![\mathrm{Int}] \mid x![\mathrm{Int}]$

race on *x*

Example: Deadlock Analysis

new y in $(x?^{s}[n].y!^{t}[n+1] \mid y?^{u}[m].x!^{v}[m+1])$

- **The type of *y*:** $s.y!^{t}[\mathrm{Int}] \mid y?^{u}[\mathrm{Int}].v$
  - **Event *s* must occur before event *u* occurs**
- **The type of *x*:** $x?^{s}[\mathrm{Int}].t \mid u.x!^{v}[\mathrm{Int}]$
  - **Event *u* must occur before event *s* occurs**

Cannot hold at once!

Example: Concurrent Objects with Non-uniform Service Availability

[Puntigam'99, Ravara&Vasconcelos.'00]

Buffer* = *put*?[*n*].*get*?[*r*].*r*![*n*].*Buffer

Can be viewed as a concurrent object with two methods ***put*** and ***get***, invoked alternately

- $\Gamma_{\mathbf{buf}} = \mu\alpha.put?[\mathrm{Int}].get?[(y)\ y![\mathrm{Int}]].\alpha$ $\vdash$ Buffer
- $\Gamma_{\mathbf{client}} = \mu\alpha.0\&(put![\mathrm{Int}] \mid get![(y)\ y![\mathrm{Int}]].\alpha)$
- $\Gamma_{\mathbf{buf}} \mid \Gamma_{\mathbf{client}}$ never get deadlocked (on outputs)
  - $\Gamma_{\mathbf{client}}$ $\vdash$ put ![2].new r in get ![r].r ?[n].P
  - $\Gamma_{\mathbf{client}}$ $\vdash$ put ![2] | new r in get ![r].r ?[n].P
  - ~~$\Gamma_{\mathbf{client}}$ $\vdash$ new r in get ![r].r ?[n].put![2]~~

General Principle Revisited

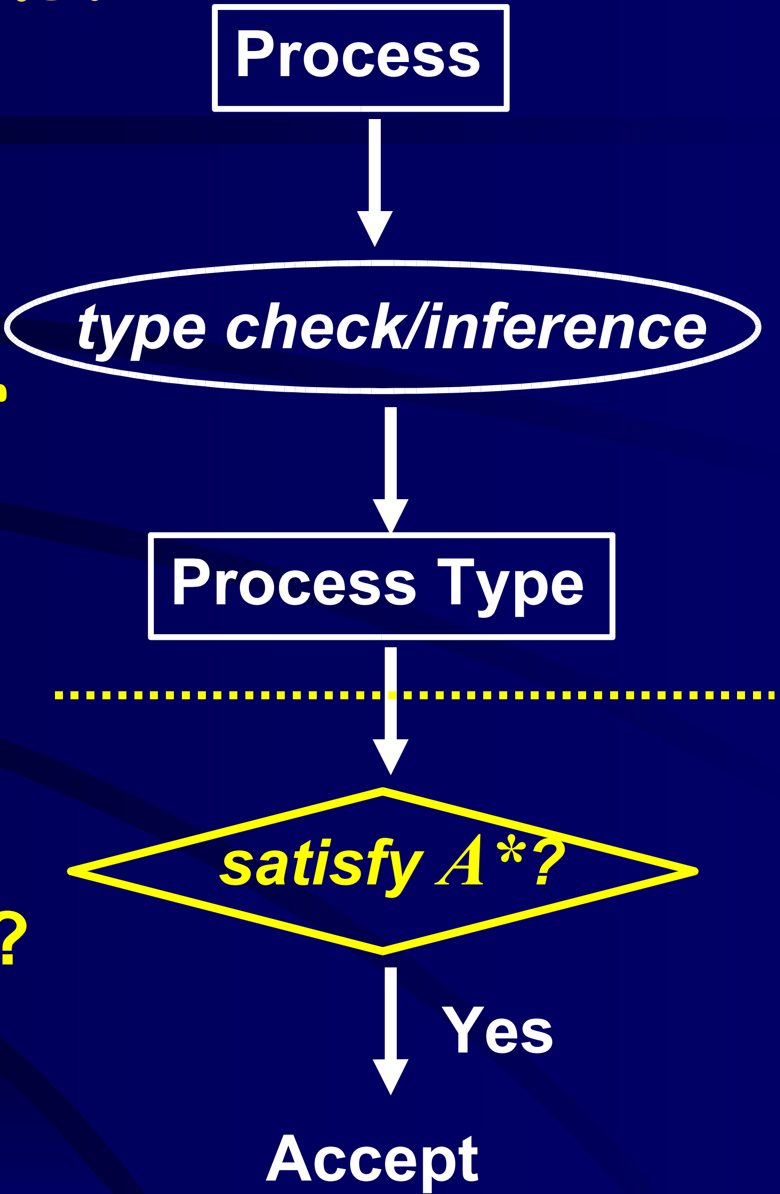
To verify a property of *P*,

verify the corresponding property of Γ

- How can we find the "corresponding" property?
- For what kind of property, does the "corresponding" property exist?

How to Obtain Specific Type Systems?

- To guarantee that every process satisfies A:
  - Choose A^* so that: *P* satisfies A whenever its type Γ satisfies A^*
  - Accept *P* only if its type Γ satisfies A^*
    - How can we find A^* ?
    - For what A, does A^* exist?



Process

type check/inference

Process Type

satisfy A^*?

Yes

Accept

Logic to Describe Properties of Processes and Types

A, B (formulae) ::=

- $x!$ (Ready to output a value on x)
- $x?$ (Ready to input a value from x)
- $A|B$ (Parallel composition of a process satisfying A and a process satisfying B)
- $\langle x \rangle A$ (Can satisfy A after a communication on x)
- $\mathrm{ev}(A)$ (Can eventually satisfy A)
- $\neg A$
- $A \vee B$
- …

Example:

$\neg \exists x.\mathrm{ev}(x! \mid x!)$ No race on output.

Semantics of the Logic

- P $\models \mathcal{A}$: Process P satisfies $\mathcal{A}$
  - $x![\ ].Q \mid y![\ ].R \models x!$
  - $x![\ ].Q \mid y![\ ].R \models x! \mid y!$
  - $x![\ ].y![\ ].Q \not\models x! \mid y!$
  - $x![\ ] \mid x?[\ ].y![\ ] \models ev(y!) \wedge \langle x \rangle y!$
- $\Gamma \models \mathcal{A}$: Process type Γ satisfies $\mathcal{A}$

Logic for Properties

$A \searrow B : \Gamma \vdash \mathsf{P}$ and $\Gamma \models A$ imply $\mathsf{P} \models B$

$A \nearrow B : \Gamma \vdash \mathsf{P}$ and $\mathsf{P} \models A$ imply $\Gamma \models B$

$$\frac{}{x? \nearrow x?} \qquad \frac{}{x! \nearrow x!} \qquad \frac{A \nearrow B}{\langle x \rangle A \nearrow \langle x \rangle B}$$

$$\frac{A \nearrow C \qquad B \nearrow D}{A|B \nearrow C|D} \qquad \frac{A \nearrow B}{\mathrm{ev}(A) \nearrow \mathrm{ev}(B)}$$

$$\frac{A \nearrow B}{\neg A \searrow \neg B} \qquad \frac{A \nearrow B}{A \nearrow B \vee C}$$

Strong Type Soundness

Theorem:

(bad) things do not happen

$A \searrow A$ for any "negative" formula A.

In other words, ...

Let A be a negative formula.

If $\Gamma \vdash P$ and $\Gamma \models A$, then $P \models A$.

……

Examples of negative formulas:

$\neg\exists x.\mathrm{ev}(x! \mid x!)$ No race on output

$\neg\exists x.\mathrm{ev}(<x>\mathrm{ev}(x! \vee x?))$ No channel is used twice

General Principle Revisited

To verify a property of P,

verify the corresponding property of Γ

- How can we find the "corresponding" property?
  Choose the property described by the same formula
- For what kind of property, does the "corresponding" property exist?

For any negative formula, at least

How to Obtain Specific Type Systems?

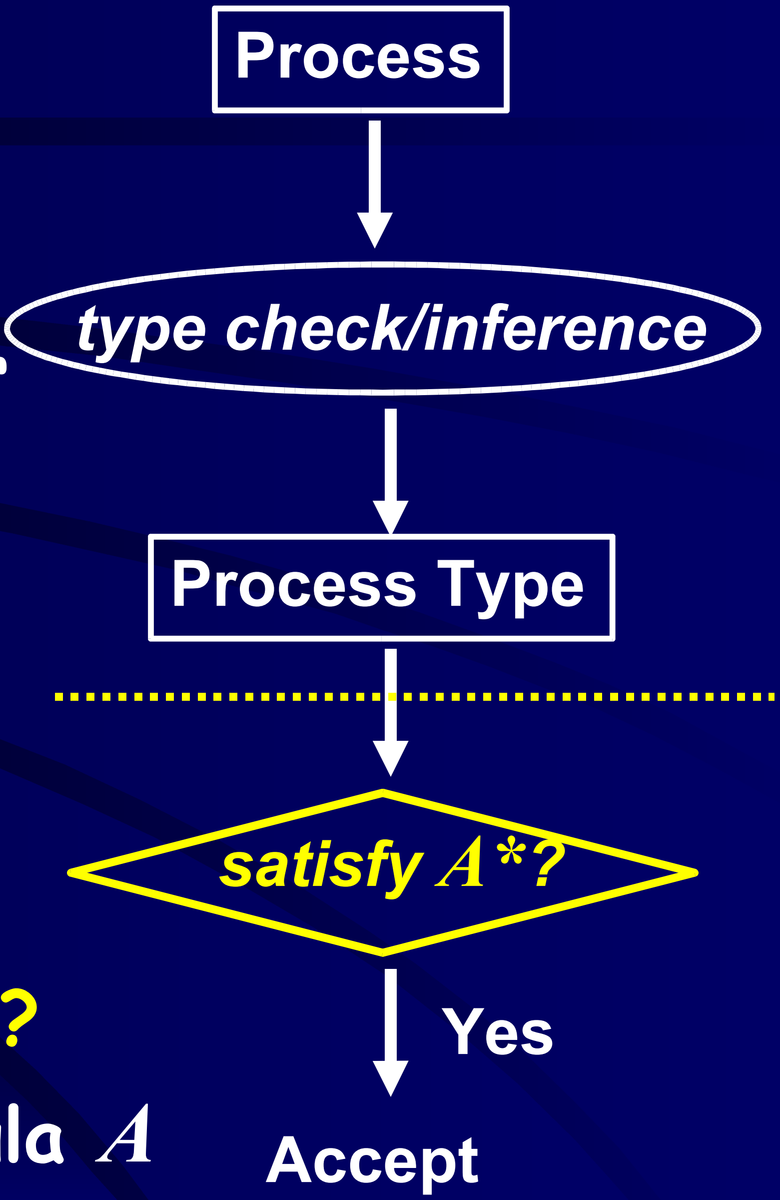
- To guarantee that every process satisfies A:
  - Choose A^* so that: **P** satisfies A whenever its type Γ satisfies A^*
  - Accept **P** only if its type Γ satisfies A^*

How can we find A^*?

$A^* = A$

For what A, does A^* exist?

For any negative formula A



Process

type check/inference

Process Type

satisfy A^*?

Yes

Accept

Deadlock Freedom Revisited

- **Deadlock freedom cannot be described by a negative formula**
  - Action labelled **t** does not deadlock = "whenever an action labelled **t** is tried, there is a further reduction"

(good) things do happen

- Separate proof of soundness required

Some Limitations

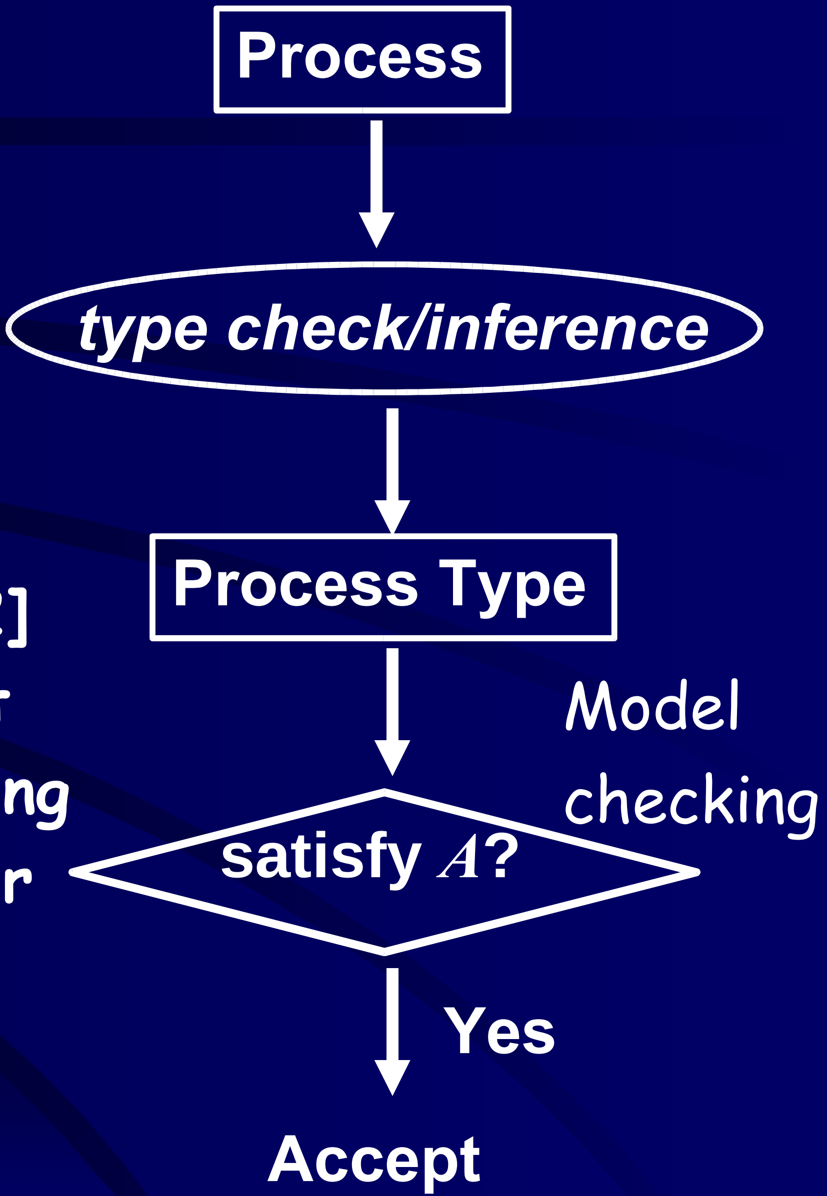
- Due to expressiveness of process types
  - Information on channel creation lost
  - Impossible to check "at most n channels are created"
- Due to the requirement for $ok(\Gamma)$
  - Invariant cond.: if $ok(\Gamma)$ and $\Gamma \to \Gamma'$, then $ok(\Gamma')$
  - Impossible to check "before x is used, y should be used"

Conclusion

Generic type system for concurrent progs

- Key idea: Abstract Processes as Types
- Many type systems are obtained as instances:
  - Race detection
  - Deadlock-freedom
    - Concurrent objects with non-uniform service availability
  - Linear channels, etc.
- Many issues can be discussed uniformly.
  - type soundness
  - type inference

- Combination of
  - Type Theory
  - Model Checking
    - Chaki et al.[POPL2002] implemented (a variant of) this framework using SPIN as model checker



Process

type check/inference

Process Type

Model checking

satisfy A?

Yes

Accept

Current/Future Work

- Extensions of the generic type system
  - More expressive power
    - Restriction operator [Chaki et al.2002, Kobayashi2005]?
  - More of common theories
    - 'Generic' typed process equivalence
- Formal verification of the correctness of the generic type system (using Coq)
  - Automatic extraction of type inference algorithms?