Substructural Type Systems for Program Analysis

Naoki Kobayashi
Tohoku University

What's This Talk About?

- A review of substructural type systems for program analysis
  - Applications
  - Common principles
    - Type Systems
    - Type Inference Algorithms
- Future directions

Outline

- Background and Motivations
  - What is type-based program analysis?
  - What are substructural type systems?
  - What are they for?
- Affine/Linear Type Systems
- Ordered Linear Type Systems
- Emerging and Future Research Topics

Type-Based Program Analysis?

- Program analysis formalized in the form of type inference
  - Types as abstract properties of a program
  - Type judgment as a relation between a program and its abstract properties
  - Type inference algorithm as an algorithm for inferring abstract properties of a program

Examples:

- type-based exception analysis
- region inference [Tofte and Talpin POPL94]
- type-based flow analysis [Palsberg POPL95]
- type-based information flow analysis [Volpano et al. 96]
- type-based deadlock analysis [Kobayashi LICS 97]

Substructural Type Systems?

- Type systems with restricted structural rules (c.f. substructural logics)

Weakening:

$$\frac{\Gamma \vdash M{:}\tau}{\Gamma, x{:}\tau' \vdash M{:}\tau}$$

Contraction:

$$\frac{\Gamma, x{:}\tau', x{:}\tau' \vdash M{:}\tau}{\Gamma, x{:}\tau' \vdash M{:}\tau}$$

Exchange:

$$\frac{\Gamma, x{:}\tau_1, y{:}\tau_2 \vdash M{:}\tau}{\Gamma, y{:}\tau_2, x{:}\tau_1 \vdash M{:}\tau}$$

Substructural Type Systems

	weakening $\dfrac{\Gamma \vdash M:\tau}{\Gamma, x:\tau' \vdash M:\tau}$	contraction $\dfrac{\Gamma, x:\tau', x:\tau' \vdash M:\tau}{\Gamma, x:\tau' \vdash M:\tau}$	exchange $\dfrac{\Gamma, x:\tau_1, y:\tau_2 \vdash M:\tau}{\Gamma, y:\tau_2, x:\tau_1 \vdash M:\tau}$
Affine	✓	✗	✓
Linear	✗	✗	✓
Ordered linear	✗	✗	✗

Substructural Type Systems

	W	C	E	Restriction on resource usage
Affine	✓	✗	✓	Can be used at most once
Linear	✗	✗	✓	Must be used exactly once
Ordered linear	✗	✗	✗	Must be used exactly once, in the specified order

Outline

- Background and Motivations
  - What is type-based program analysis?
  - What are substructural type systems?
  - What are they for?
- Affine/Linear Type Systems
- Ordered Linear Type Systems
- Future Directions

Why Affine Types?
(why "at most once" condition?)

- Memory management [Baker, "Linear LISP"]
  - Memory space for an affine value can be deallocated after the value is used.
- Optimization
  - Inlining (for lazy languages) [Turner et al. FPCA95]

    $\text{let } x = M \text{ in } N \Rightarrow [M/x]N \quad (\text{if } x \text{ is affine})$
  - One-shot call/cc
  - "tail-call optimization" for message-passing programs
- Security
  - Nonce should not be used twice
    [Abadi, "secrecy by typng"]
  - Linear declassification (e.g. password check)
    [Kaneko&Kobayashi, ESOP 2008]

Why Linear Types?
(why "exactly once" condition?)

- Finalization of resource
  - A memory cell should be eventually deallocated.
  - A file should be eventually closed.

- Synchronization/communication protocols
  - An acquired lock should be eventually released.
  - A server should send a reply to each request exactly once.

Why Ordered Types?

- Checking resource access protocols [Igarashi&Kobayashi, POPL2002]
  - An array should be initialized before being read.
  - A memory cell must not be read after deallocation
  - A file must not be read/written after being closed.
- Preventing deadlock [Kobayashi 97-]
- Streaming XML processing [Suenaga et al. 04]
  - Tree data in streams can be accessed only in a restricted order.

Outline

- Background and Motivations
- Affine/Linear Type Systems
  - λ-calculus with affine/linear resources
  - Type systems
  - Type inference algorithms
- Ordered Linear Type Systems
- Future Directions

λ-calculus with resource

M (term) ::= x | c | $\lambda x.M$ | $M_1 M_2$

| if M_1 then M_2 else M_3 | let x = M_1 in M_2

| new( ) resource creation

| use(M) resource access

Semantics

- Run-time state: (H, M)

  H ∈ Resource → {0, 1}

- Reduction

  (H, E[new( )]) → (H{r:1}, E[r]) (r is fresh)

  (H{r:1}, E[use r]) → (H{r:0}, E[()])

  (H{r:0}, E[use r]) → Error

E.g. ({}, let y= new( ) in (use y; use y))

→ ({r:1}, let y=r in (use y; use y))

→ ({r:1}, use r; use r)

→ ({r:0}, use r)

→ Error

Functions as Resources

$$\text{fun } x \Rightarrow M \equiv (\lambda x.M, \text{ new}(\,))$$

$$\text{app}(M_1, M_2) \equiv \text{let } x{=}M1 \text{ in let } y{=}M2 \text{ in use}(\text{snd}(x));\ (\text{fst } x)(y)$$

M (term) ::= $x \mid c \mid \lambda x.M \mid M_1M_2$

| if M_1 then M_2 else M_3 | let $x = M_1$ in M_2

| new( ) resource creation

| use M resource access

Expected Properties

- Affine type system:
  If M is well-typed, then:
  $(\{\}, M) \not\to^* \mathbf{Error}$
  (No resource can be used twice)

- Linear type system:
  If M is well-typed, then:
  (i) $(\{\}, M) \not\to^* \mathbf{Error}$
  (ii) $(\{\}, M) \to^* (H, c)$ implies
  $H(r)=0$ for every $r \in dom(H)$
  (Every resource is used)

Types

τ (types)	::=	b	base types
	|	R(u)	resource types
	|	$(\tau \to \tau$, u)	function types
	|	$\tau \times \tau$	pair types
u (uses)	::=	0	cannot be used
	|	1	exactly once (linear type only)
	|	$\leq$1	at most once (affine type only)
	|	ω	any number of times

Type Judgment (examples)

- ✓ x: R(1) ⊢ use(x): unit
- ✗ x: R(1) ⊢ use(x); use(x): unit
- ✓ x: R(ω) ⊢ use(x); use(x): unit
- ✗ x: R(1) ⊢ ( ): unit
- ✓ x: R($\leq$1) ⊢ ( ): unit
- ✓ x: R(1) ⊢ λy.use(x): (unit $\rightarrow$ unit, 1)
- ✗ x: R(1) ⊢ λy.use(x): (unit $\rightarrow$ unit, ω)

Typing (structural rules)

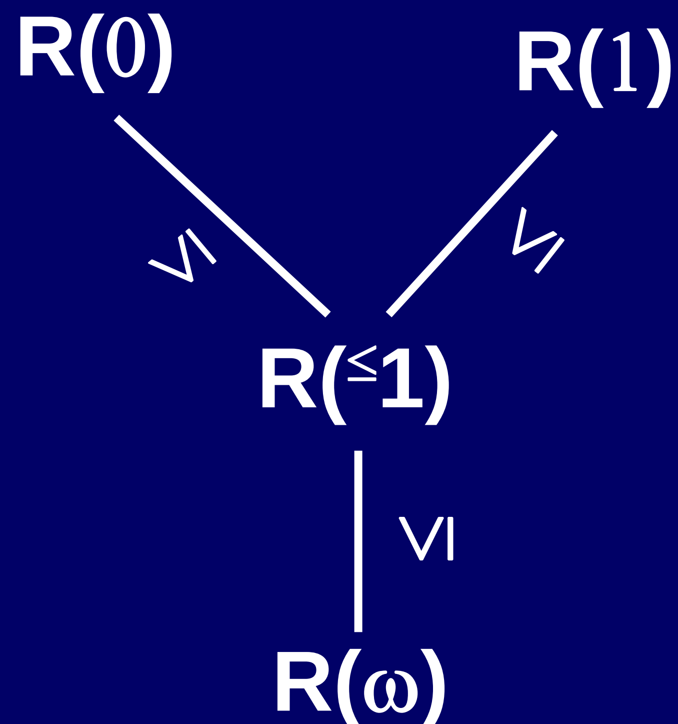
$$\frac{\Gamma, x{:}\tau_1, y{:}\tau_2, \Delta \vdash M{:}\sigma}{\Gamma, y{:}\tau_2, x{:}\tau_1, \Delta \vdash M{:}\sigma}\ \text{(exchange)}$$

$$\frac{\Gamma \vdash M{:}\sigma \quad \text{nonlinear}(\tau)}{\Gamma, x{:}\tau \vdash M{:}\sigma}\ \text{(weakening)}$$

$$\checkmark\ \frac{x{:}R(1) \vdash use(x){:}unit}{x{:}R(1), y{:}R(\leq 1) \vdash use(x){:}unit} \qquad \times\ \frac{x{:}\ R(1) \vdash use(x){:}unit}{x{:}R(1),\ y{:}R(1) \vdash use(x){:}unit}$$

Typing: subsumption

$$\frac{\Gamma \vdash M:\tau \qquad \tau \le \sigma}{\Gamma \vdash M:\sigma}\ \text{(subsumption)}$$



Typing for resources

$$\frac{}{\vdash \mathit{newA}(\ ): R(\leq 1)}\ \text{(affine resource)}$$

$$\frac{}{\vdash \mathit{newL}(\ ): R(1)}\ \text{(linear resource)}$$

$$\frac{\Gamma \vdash M: R(1)}{\Gamma \vdash \text{use } M: \text{unit}}$$

Typing for resources

$$\frac{}{\Gamma \vdash \mathsf{newA}(\ ): R(\leq 1)}\ \text{(affine resource)}$$

$$\frac{}{\vdash \mathsf{newL}(\ ): R(1)}\ \text{(linear resource)}$$

$$\frac{\Gamma \vdash M: R(1)}{\Gamma \vdash \mathsf{use}\ M: \mathsf{unit}}$$

Typing : let

$$\frac{\Gamma \vdash M:\tau \qquad \Delta, x:\tau \vdash N:\sigma}{\Gamma + \Delta \vdash \text{let } x=M \text{ in } N : \sigma}$$

Example:

$$\frac{r:R(1) \vdash use(r):unit \qquad r:R(1),\ x:unit \vdash use(r):unit}{r:R(1)+R(1) \vdash \text{let } x=use(r) \text{ in } use(r) : unit}$$

R(u) + R(u') = R(u+u') where:

+	0	1	ω
0	0	1	ω
1	1	ω	ω
ω	ω	ω	ω

Typing : let

$$\frac{\Gamma \vdash M:\tau \qquad \Delta, x:\tau \vdash N:\sigma}{\Gamma + \Delta \vdash \mathtt{let}\ x{=}M\ \mathtt{in}\ N : \sigma}$$

Example:

$$\frac{r{:}R(1) \vdash use(r){:}unit \qquad r{:}R(1),\ x{:}unit \vdash use(r){:}unit}{r{:}R(\omega) \vdash \mathtt{let}\ x{=}use(r)\ \mathtt{in}\ use(r) : unit}$$

$R(u) + R(u') = R(u+u')$ where:

+	0	1	ω
0	0	1	ω
1	1	ω	ω
ω	ω	ω	ω

Typing : let

$$\cancel{\dfrac{\Gamma \vdash M:\tau \qquad \Gamma, x:\tau \vdash N:\sigma}{\Gamma \vdash \text{let } x=M \text{ in } N : \sigma}}$$

Example:

$$\frac{r:R(1) \vdash use(r):unit \qquad r:R(1),\ x:unit \vdash use(r):unit}{r:R(\omega) \vdash \text{let } x=use(r) \text{ in } use(r) : unit}$$

R(u) + R(u') = R(u+u') where:

+	0	1	ω
0	0	1	ω
1	1	ω	ω
ω	ω	ω	ω

Outline

- Background and Motivations
- Affine/Linear Type Systems
  - λ-calculus with affine/linear resources
  - Type systems
  - Type inference algorithms
    - polynomial-time algorithm for affine types
    - NP-completeness of linear type system
    - tractable linear type systems
- Ordered Linear Type Systems
- Future Directions

Type Inference For Linear/Affine Type Systems

- Prepare variables to denote unknown uses
- Extract subtype constraints

  $\tau_1 \leq \sigma_1, \ldots, \tau_n \leq \sigma_n$

- Reduce subtype constraints to constraints on use variables

  $\eta_1 \leq u_1, \ldots, \eta_n \leq u_n$

- Solve subuse constraints

Affine Type Inference: Example

```
let rec f(n, x) =
   if n=0 then use(x)
   else f(n-1, x)
in
let r = newA()
in f(3, r)
```

Affine Type Inference: Example

```
let rec f(n, x: R(η) ) =
   if n=0 then use(x)
   else f(n-1, x)
in
let r = newA()
in f(3, r)
```

Affine Type Inference: Example

```
let rec f(n, x: R(η) ) =
    if n=0 then use(x)
    else f(n-1, x)
in
let r = newA()
in f(3, r)
```

$R(\eta) \leq R(^{\leq}1)$

$R(\omega) \leq R(^{\leq}1) \leq R(0)$

Affine Type Inference: Example

```
let rec f(n, x: R(η) ) =
   if n=0 then use(x)
   else f(n-1, x)
in
let r = newA()
in f(3, r)
```

$R(\eta) \leq R(^{\leq}1)$

$R(\eta) \leq R(\eta)$

$R(\omega) \leq R(^{\leq}1) \leq R(0)$

Affine Type Inference: Example

```
let rec f(n, x: R(η) ) =
    if n=0 then use(x)
    else f(n-1, x)
in
let r = newA()
in f(3, r)
```

$R(\eta) \leq R(^{\leq}1)$

$R(\eta) \leq R(\eta)$

$R(^{\leq}1) \leq R(\eta)$

$R(\omega) \leq R(^{\leq}1) \leq R(0)$

Affine Type Inference: Example

```
let rec f(n, x: R(η) ) =
    if n=0 then use(x)
    else f(n-1, x)
in
let r = newA()
in f(3, r)
```

$R(\eta) \leq R(^{\leq}1)$

$R(\eta) \leq R(\eta)$

$R(^{\leq}1) \leq R(\eta)$

$\downarrow$

$\eta \leq {}^{\leq}1,\ \eta \leq \eta,\ {}^{\leq}1 \leq \eta$

$(\omega \leq {}^{\leq}1 \leq 0)$

$\downarrow$

$\eta = {}^{\leq}1$

Constraint solving for uses: Affine case

$$\begin{aligned}
\eta_1 &\leq f_1(\eta_1, \ldots, \eta_n) \\
&\ldots \\
\eta_n &\leq f_n(\eta_1, \ldots, \eta_n) \\
{}^{\leq}1 &\leq g_1(\eta_1, \ldots, \eta_n) \\
&\ldots \\
{}^{\leq}1 &\leq g_k(\eta_1, \ldots, \eta_n)
\end{aligned}$$

$\eta_1, \ldots, \eta_n$: **use variables**

$f_1, \ldots, f_n, g_1, \ldots, g_k$:

monotonic functions

(constructed from +, $\times$, lub, 0, ${}^{\leq}1$)

Constraint solving for uses: Affine case

$$\eta_1 \leq f_1(\eta_1, \ldots, \eta_n)$$
$$\ldots$$
$$\eta_n \leq f_n(\eta_1, \ldots, \eta_n)$$

$$^{\leq}1 \leq g_1(\eta_1, \ldots, \eta_n)$$
$$\ldots$$
$$^{\leq}1 \leq g_k(\eta_1, \ldots, \eta_n)$$

$\eta_1, \ldots, \eta_n$: use variables

$f_1, \ldots, f_n, g_1, \ldots, g_k$:
monotonic functions
(constructed from +, $\times$, lub, 0, $^{\leq}1$)

1. Use a fixedpoint computation algorithm to get the greatest solution $\vec{\eta} = \vec{c}$ for $\vec{\eta} \leq \vec{f}(\vec{\eta})$
   (n.b. $\vec{0} \geq \vec{f}(\vec{0}) \geq \vec{f}(\vec{f}(\vec{0})) \geq \ldots$)

Constraint solving for uses: Affine case

$\eta_1 \leq f_1(\eta_1, \ldots, \eta_n)$
$\ldots$
$\eta_n \leq f_n(\eta_1, \ldots, \eta_n)$

${}^{\leq}1 \leq g_1(\eta_1, \ldots, \eta_n)$
$\ldots$
${}^{\leq}1 \leq g_k(\eta_1, \ldots, \eta_n)$

$\eta_1, \ldots, \eta_n$: use variables

$f_1, \ldots, f_n, g_1, \ldots, g_k$:
monotonic functions
(constructed from $+$, $\times$, lub, 0, ${}^{\leq}1$)

1. Use a fixedpoint computation algorithm to get the greatest solution $\vec{\eta} = \vec{c}$ for $\vec{\eta} \leq \vec{f}(\vec{\eta})$
   (n.b. $\vec{0} \geq \vec{f}(\vec{0}) \geq \vec{f}(\vec{f}(\vec{0})) \geq \ldots$)
2. Check $\vec{{}^{\leq}1} \leq \vec{g}(\vec{c})$

Linear in the size of the constraints [Rehof&Mogensen, 1999]

Constraint solving for uses:
Linear case

$$\eta_1 \leq f_1(\eta_1, \ldots, \eta_n)$$

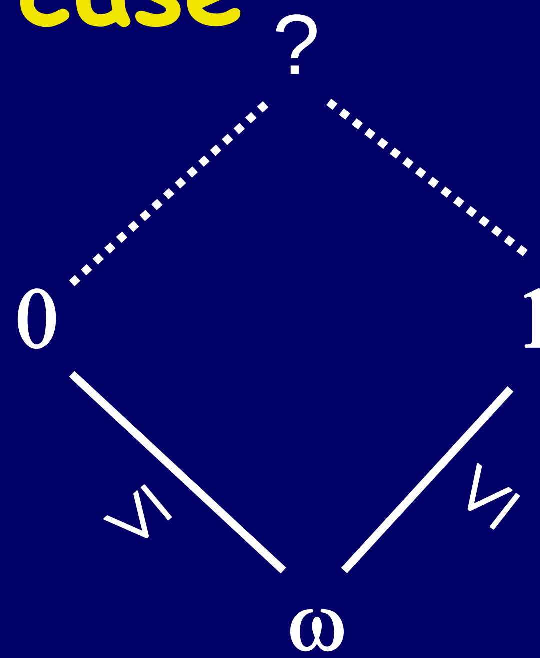
$$\ldots$$

$$\eta_n \leq f_n(\eta_1, \ldots, \eta_n)$$

$$1 \leq g_1(\eta_1, \ldots, \eta_n)$$

$$\ldots$$

$$1 \leq g_k(\eta_1, \ldots, \eta_n)$$



The same algorithm does NOT apply!

Linear type system is NP-complete!

- 1-in-3SAT problem can be encoded.

$\oplus(X, Y, \neg Z) \wedge \oplus(\neg X, \neg Y, Z)$

$\oplus(A, B, C)$: Exactly one of A, B, C is true

iff

$f_X: R(\eta_X) \to \mathrm{unit},\ \ f_{\neg X}: R(\eta_{\neg X}) \to \mathrm{unit},$
$f_Y: R(\eta_Y) \to \mathrm{unit},\ \ f_{\neg Y}: R(\eta_{\neg Y}) \to \mathrm{unit},$
$f_Z: R(\eta_Z) \to \mathrm{unit},\ \ f_{\neg Z}: R(\eta_{\neg Z}) \to \mathrm{unit}$

$\vdash$

let r=newL() in $(f_X(r);\ f_{\neg X}(r))$; $\qquad \eta_X + \eta_{\neg X} = 1$
let r=newL() in $(f_Y(r);\ f_{\neg Y}(r))$; $\qquad \eta_Y + \eta_{\neg Y} = 1$
let r=newL() in $(f_Z(r);\ f_{\neg Z}(r))$; $\qquad \eta_Z + \eta_{\neg Z} = 1$
let r=newL() in $(f_X(r);\ f_Y(r);\ f_{\neg Z}(r))$; $\qquad \eta_X + \eta_Y + \eta_{\neg Z} = 1$
let r=newL() in $(f_{\neg X}(r);\ f_{\neg Y}(r);\ f_Z(r))$ $\qquad \eta_{\neg X} + \eta_{\neg Y} + \eta_Z = 1$

Linear type system is NP-complete!

- 1-in-3SAT problem can be encoded.

$\oplus(X, Y, \neg Z) \wedge \oplus(\neg X, \neg Y, Z)$

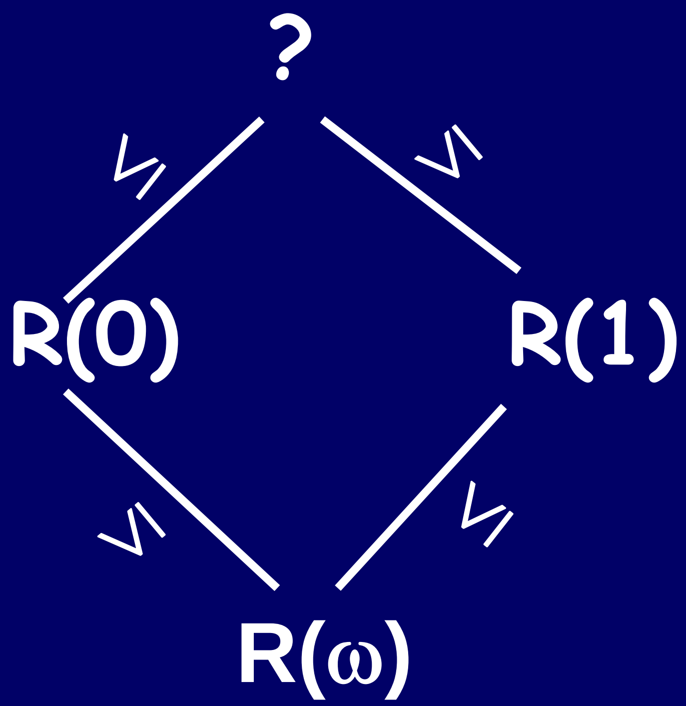
iff

$\oplus(A, B, C)$: Exactly one of A, B, C is true

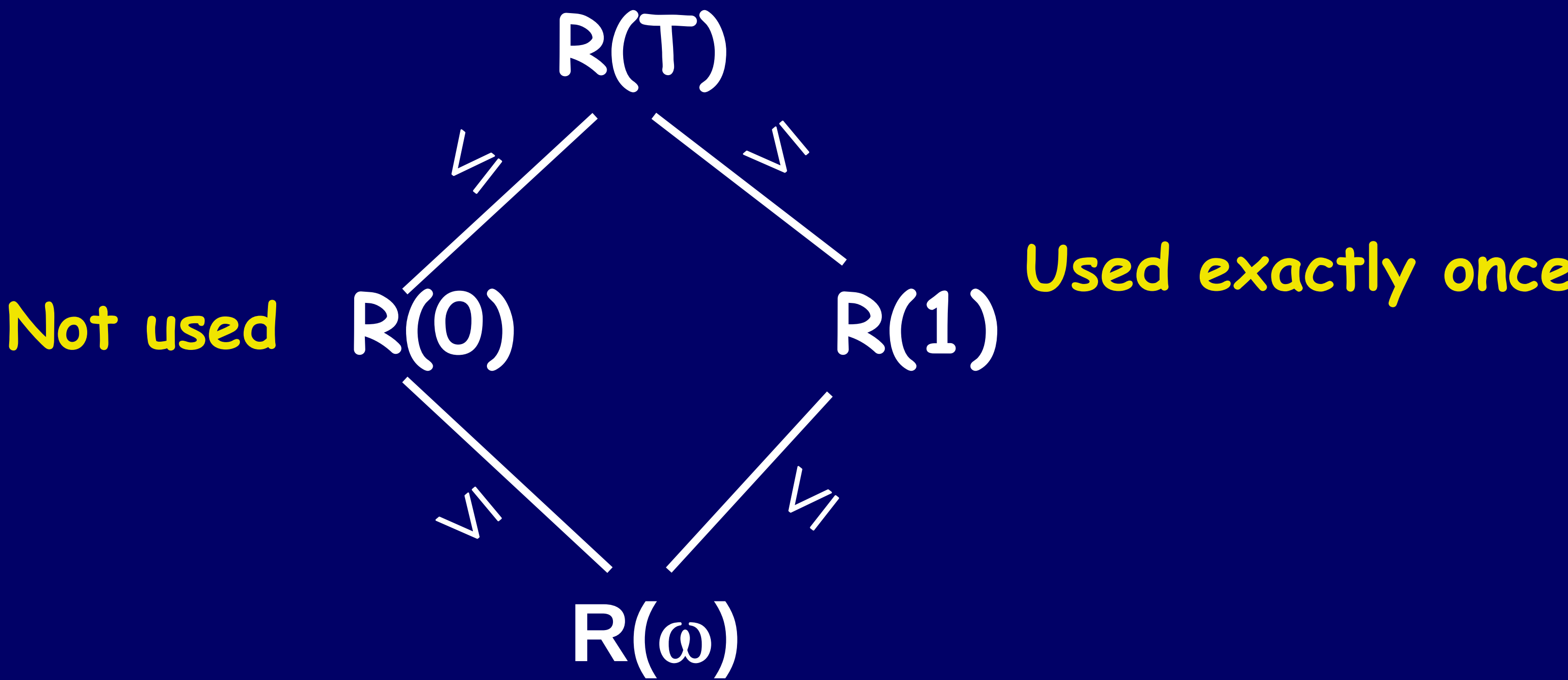
$\vdash$

let $f_X(r) = f_X(r)$ in let $f_{\neg X}(r) = f_{\neg X}(r)$ in
let $f_Y(r) = f_Y(r)$ in let $f_{\neg Y}(r) = f_{\neg Y}(r)$ in
let $f_Z(r) = f_Z(r)$ in let $f_{\neg Z}(r) = f_{\neg Z}(r)$ in
let r = newL() in $(f_X(r)\ ;\ f_{\neg X}(r))$;
let r = newL() in $(f_Y(r)\ ;\ f_{\neg Y}(r))$;
let r = newL() in $(f_Z(r)\ ;\ f_{\neg Z}(r))$;
let r = newL() in $(f_X(r)\ ;\ f_Y(r);\ f_{\neg Z}(r))$;
let r = newL() in $(f_{\neg X}(r)\ ;\ f_{\neg Y}(r);\ f_Z(r))$

Tractable Linear Type System

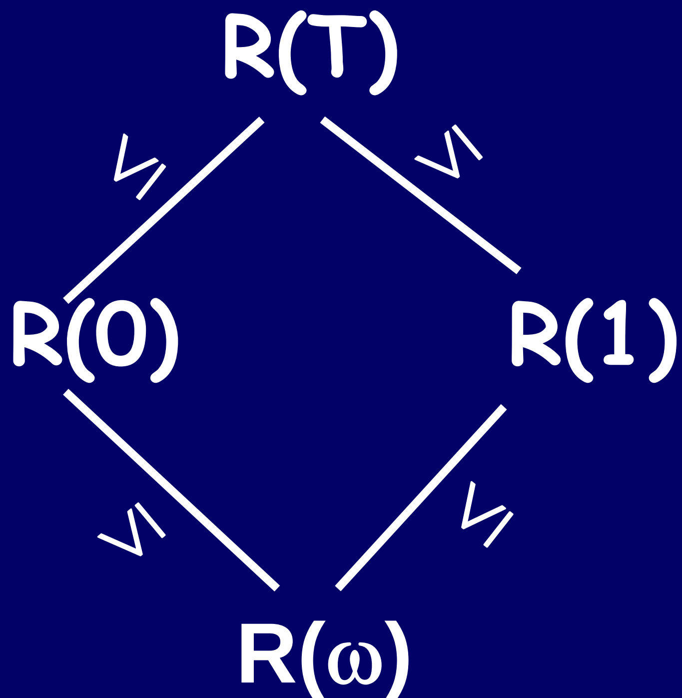


Tractable Linear Type System



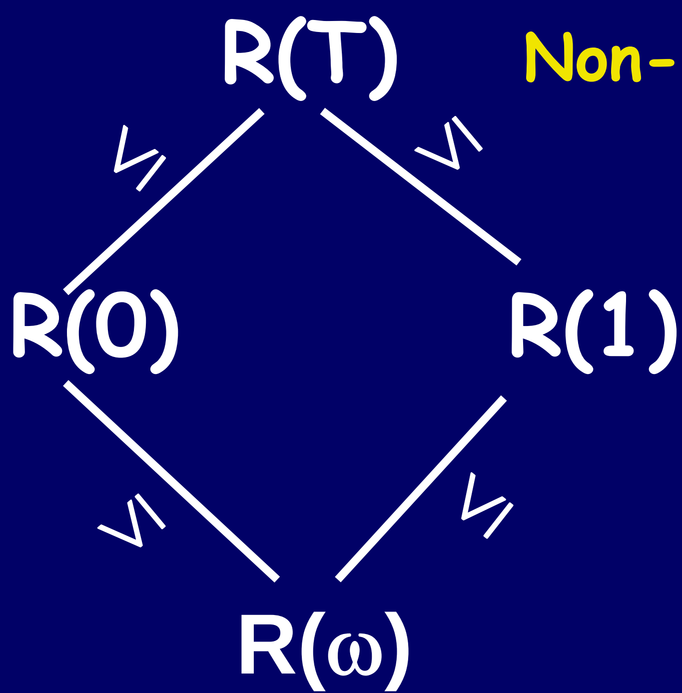
Tractable Linear Type System

Not used



Used exactly once
if the program terminates.
Otherwise, at most once.

Tractable Linear Type System



Non-termination with no use

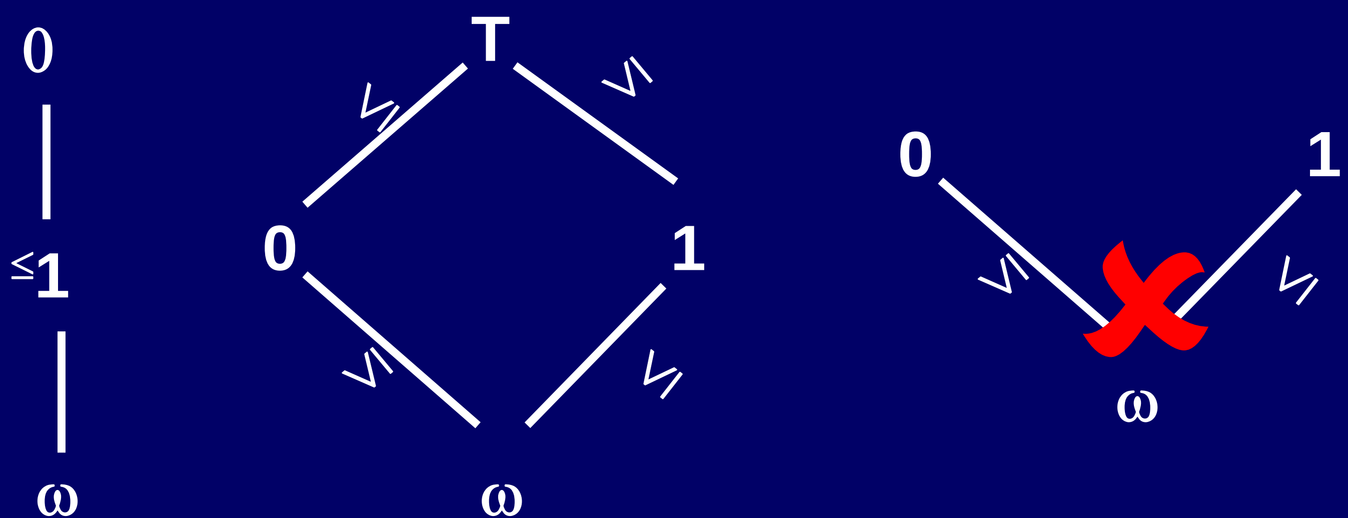
Not used

Used exactly once
if the program terminates.
Otherwise, at most once.

> Rehof&Mogensen's algorithm
> is applicable!

Affine/Linear Types: Lessons

- Extend resource types with uses
- Extend also function types with uses
- Carefully restrict structural rules
- Carefully design the domain of uses (to enable efficient type inference)



Outline

- ♦ Background and Motivations
- ♦ Affine/Linear Type Systems
- ♦ Ordered Linear Type Systems
  - – λ-calculus with order-constrained resources
  - – Type system
  - – Type inference
- ♦ Emerging and Future Research Topics

λ-calculus with ordered resource [Igarashi&Kobayashi, POPL02]

M (term) ::= x | c | $\lambda x.M$ | $M_1 M_2$

| if M_1 then M_2 else M_3 | let x = M_1 in M_2

| $\mathbf{new}^{\Phi}()$ creation of resource used according to Φ

| $\mathbf{use}_a(M)$ resource access

Φ : A set of valid access sequences

Example

Should be closed after some read operations

✓ let fp = new$^{r^*c}$() in
read(fp); close(fp)

✗ let fp = new$^{r^*c}$() in
close(fp) ; read(fp)

✗ let fp = new$^{r^*c}$() in
if b then read(fp) else close(fp)

(read, write, close as abbreviations for use_r, use_w, use_c)

Semantics

- Reduction

  $(H, E[\mathrm{new}^{\Phi}()]) \to (H\{r: \Phi\}, E[r])$ (r is fresh)

  $(H\{r: \Phi\}, E[\mathrm{use}_a\ r]) \to (H\{r: \{w \mid aw \in \Phi\}\}, E[()])$

  $(H\{r: \Phi\}, E[\mathrm{use}_a\ r]) \to$ Error

  (if $\{w \mid aw \in \Phi\} = \{\ \}$)

E.g. $(\{\}, \text{let } y=\mathrm{new}^{r^*c}() \text{ in } (\mathrm{use}_C\ y; \mathrm{use}_R\ y))$

$\to (\{x: R^*C\}, \text{let } y=x \text{ in } (\mathrm{use}_C\ y; \mathrm{use}_R\ y))$

$\to (\{x: R^*C\}, \mathrm{use}_C\ x; \mathrm{use}_R\ x)$

$\to (\{x: \{\varepsilon\}\}, \mathrm{use}_R\ x)$

$\to$ Error

Expected Properties

- If M is well-typed, then:

  (i) $(\{\}, M) \not\rightarrow^* \text{Error}$
  (no invalid access)

  (ii) $(\{\}, M) \rightarrow^* (H, c)$ implies
  $\varepsilon \in H(r)$ for every $r \in dom(H)$
  (finalization)

Types

τ (types)	::=	b	base types
	|	R(u)	resource types
	|	$\tau_1 \rightarrow \tau_2$	function types
	|	$\tau \times \tau$	
u (usages)	::=	0	cannot be used
	|	a	accessed once by use_a
	|	$u_1; u_2$	u_1 and then u_2
	|	$u_1 \& u_2$	u_1 or u_2
	|	ρ	usage variable
	|	$\mu\rho.\, u$	recursion

Examples: usages

- $\mu\rho.(c \mathbin{\&} (r; \rho))$: read-only file

- $\mu\rho.(0 \mathbin{\&} (push;\rho; pop))$: stack

u (usages) ::=		0	cannot be used
	|	a	accessed once by use_a
	|	$u_1; u_2$	u_1 and then u_2
	|	$u_1 \& u_2$	u_1 or u_2
	|	ρ	usage variable
	|	$\mu\rho.\,u$	recursion

Typing : let

$$\frac{\Gamma \vdash M:\tau \qquad \Delta, x:\tau \vdash N:\sigma \qquad rfree(\tau)}{\Gamma ; \Delta \vdash \text{let } x=M \text{ in } N : \sigma}$$

(rfree(τ) if τ does not contain resource types)

Example:

$$\frac{y{:}\ R(r) \vdash read(y){:}unit \qquad y{:}\ R(c),\ x{:}\ unit \vdash close(y){:}unit}{y{:}\ R(r;c) \vdash \text{let } x = read(y) \text{ in } close(y) : unit}$$

Type Inference: Example

```
let rec f(n, x) =
  if n=0  then close(x)
  else (read(x);f(n-1, x))
in
let r = new^{r*c}()
in f(3, r)
```

Type Inference: Example

```
let rec f(n, x: R(ρ) ) =
  if n=0  then close(x)
  else (read(x);f(n-1, x))
in
let r: R(η) = new^{r*c}()
in f(3, r)
```

Type Inference: Example

```
let rec f(n, x: R(ρ) ) =
  if n=0  then close(x)
  else (read(x);f(n-1, x))
in
let r: R(η) = new^{r*c}()
in f(3, r)
```

$R(\rho) \leq R(c)$

$R(\rho) \leq R(r);\ R(\rho)$

$R(\eta) \leq R(\rho)$

$sem(\eta) \subseteq r^*c$

↓

$\rho \leq c\ \&\ (r;\ \rho)$

$\eta \leq \rho$

$sem(\eta) \subseteq r^*c$

↓

✓ $sem(\mu r.c\ \&\ (r;\ \rho)) \subseteq r^*c$

Outline

- Background and Motivations
- Affine/Linear Type Systems
- Ordered Linear Type Systems
- Emerging and Future Directions
  - Fractional Types
  - Ordered Linear Datatypes
  - Better Ordered Type Systems
  - Integration with Other Verification Methods

Fractional Types

Type of resource that can be used 0.5 times

```
fun f(x: R(≤0.5), y: R(≤0.5)) =
                    if x=y then use(x) else ()
```

What are they for?

- More expressive power
- Efficient type inference (via linear programming)

for

- Race analysis [Boyland, SAS03] [Terauchi,CONCUR06, etc.]
- Protocol verification [Kikuchi & Kobayashi, APLAS2007]

Ordered Pair Types

$\tau \otimes \sigma$:

Type of a pair of values of types τ and σ with no order constraint

$\tau \blacktriangleright \sigma$

Type of pair (v,w) where v is used according to τ and then w is used according to σ

$\tau \blacktriangleleft \sigma$

Type of pair (v,w) where w is used according to σ and then v is used according to τ

Ordered List/Tree Types

$\mu\,\alpha.\,(\mathrm{unit} + \tau \blacktriangleright \alpha)$:

A list accessed from the head

$\mu\,\alpha.\,(\mathrm{unit} + \tau \blacktriangleleft \alpha)$:

A list accessed from the tail

$\mu\,\alpha.\,(\mathrm{unit} + (\alpha \blacktriangleleft \tau) \blacktriangleright \alpha)$:

A tree accessed in the depth-first, left-to-right order

Application: Stream processing of XML [Suenaga et al. 2004]

Better Ordered Type Systems?

- Naive rule is unsound

$$\frac{\Gamma \vdash M{:}\tau \qquad \Delta, x{:}\tau \vdash N{:}\sigma \qquad \cancel{\mathrm{rfree}(\tau)}}{\Gamma ; \Delta \vdash \mathrm{let}\ x{=}M\ \mathrm{in}\ N : \sigma}$$

$$\frac{y{:}R(r) \vdash y{:}R(r) \qquad y{:}R(c),\ x{:}R(r) \vdash \mathrm{close}(y);\mathrm{read}(x){:}\ \mathrm{unit}}{y{:}\ R(r;c) \vdash \mathrm{let}\ x{=}\ y\ \mathrm{in}\ \mathrm{close}(y);\ \mathrm{read}(x) : \mathrm{unit}}$$

Better Ordered Type Systems?

- Naive rule is unsound

$$\frac{\Gamma \vdash M:\tau \qquad \Delta, x:\tau \vdash N:\sigma}{\Gamma ; \Delta \vdash \text{let } x=M \text{ in } N : \sigma}$$

- Existing solutions
  - Restrict types ("rfree" condition) ([Suenaga et al. 2004] for XML processing)
  - Introduce temporal operators ([Igarashi&Kobayashi 2002], for resource usage analysis)
  - Introduce "levels" to express causal dependencies ([Kobayashi 97, for deadlock analysis)

Integration with Other Verification Methods?

- Need for value-dependent information

```
let
    x = if y>0 then newL() else null
in
    if y>0 then use(x) else ( )
```

Substructural Type Systems: Summary

- Useful for checking resource usage
- Must be carefully designed to ensure:
  - Type soundness
  - Efficient type inference

    Often reduced to:
    - Fixedpoint computation for monotonic functions
    - Language inclusion problem (e.g. CFL vs RL)
    - Model checking problem

Emerging and Future Topics

- Fractional types
  - utilization of linear programming
- Ordered linear datatypes
  - More applications?
- Better ordered type systems
- Integration with other verification methods