Linear Type Systems for Concurrent Languages

Eijiro Sumii
Naoki Kobayashi

University of Tokyo

Merit and Demerit of Concurrent Languages

Compared with sequential languages

- Merit: more expressive power
  - Inherently concurrent application (e.g. GUI)
  - Parallel/distributed computation
- Demerit: more complicated behavior
  - Non-determinism (possibility of various results)
  - Deadlock (failure of due communication)

➡ **Errors & Inefficiencies**

Example of Complication (1)

In ML:

```
r:int ref ⊢
 (r:=3; r:=7; !r)
```
(evaluates to* **7** **)*

In CML [Reppy 91]:

```
c:int chan ⊢
 (spawn(fn()=>send(c, 3));
  spawn(fn()=>send(c, 7));
  recv(c))
```
(evaluates to either* **3** *or* **7** **)*

Example of Complication (2)

In ML:

```
let val r:int ref = ref 3
in !r + !r + !r
end (* evaluates to 9 *)
```

In CML:

```
let val c:int chan = channel()
    val _ = spawn(fn()=>send(c, 3))
in recv(c) + recv(c) + recv(c))
end (* evaluation gets stuck! *)
```

Example of Complication (3)

In ML:

```
let val r1:bool ref = ref false
    val r2:bool ref = ref true
in !r2 andalso !r1
end (* evaluates to false *)
```

In CML:

...

Example of Complication (3)

In ML:

...

In CML:

```
let val c1:bool chan = channel ()
    val c2:bool chan = channel ()
    val _ = spawn(fn()=>
                       (send(c1, false);
                        send(c2, true)))
in recv(c2) andalso recv(c1)
end (* evaluation gets stuck! *)
```

Our Approach

Identify deterministic/deadlock-free parts by a static type system

Enrich Channel Types with Information of

- "In what way a channel is used"

  ⇒ Linear Channels, Usage Annotations

- "In what order channels are used"

  ⇒ Time Tags

Rationale:

Type systems are usually compositional & tractable

(unlike model checking, abstract interpretation, etc.)

Outline

- Introduction
- Basic Ideas
  - Linear Channels [Kobayashi et al. 96]
  - Time Tags [Kobayashi 97]
- Formalization in process calculi
- Extension by Usage Annotations [Sumii & Kobayashi 98]
- Conclusion

Basic Ideas (1): Linear Channels

c : p^m t **chan**

p (polarity) ::= - (output) | ¯ (input) | ↕ (both) | ½ (none)

"<u>In which direction</u>* c *can be used"

m (multiplicity) ::= **1** (exactly once) | w (any times)

"<u>How many times</u>* c *can be used"

✓ `c:-`^{1} `int chan` ⊢ `send(c, 3):unit`

✗ `c:-`^{1} `int chan` ⊢ `recv(c):int`

Basic Ideas (1): Linear Channels

c : p^{m} t chan

p (polarity) ::= - (output) | ¯ (input) | ↕ (both) | ½ (none)

*"<u>In which direction</u> **c** can be used"*

m (multiplicity) ::= **1** (exactly once) | w (any times)

"<u>How many times</u> c can be used"

✓ **c:$-^{1}$ int chan ⊢ send(c, 3):unit**

✗ **c:$-^{1}$ int chan ⊢ (send(c, 3); send(c, 7)):unit**

Basic Ideas (1): Linear Channels

$$\texttt{c:}{-}^{1}\ \texttt{int chan} \vdash \texttt{send(c, 3):unit}$$

$$\texttt{c:}\overline{}^{1}\ \texttt{int chan} \vdash \texttt{recv(c):int}$$

$$\texttt{c:}\updownarrow^{1}\ \texttt{int chan} \vdash$$

```
(spawn(fn()=>send(c, 3));
 recv(c)):int
```

Basic Ideas (2): Time Tags

✓ $\mathtt{c}: \bar{\ }_{s}^{1}$ `bool chan,`
$\mathtt{d}: \bar{\ }_{t}^{1}$ `bool chan;` $s \prec t \vdash$
`(recv(c) andalso recv(d)):bool`

✗ $\mathtt{c}: \bar{\ }_{s}^{1}$ `bool chan,`
$\mathtt{d}: \bar{\ }_{t}^{1}$ `bool chan;` $s \prec t \vdash$
`(recv(d) andalso recv(c)):bool`

Basic Ideas (2): Time Tags

✗ c: $^{-}{}_{s}^{1}$ bool chan,
  d: $^{-}{}_{t}^{1}$ bool chan; $s \prec t \vdash$
  (spawn(fn()=>(send(c, true);
                send(d, false)));
   recv(d) andalso recv(c)):bool

✗ c: $^{-}{}_{s}^{1}$ bool chan,
  d: $^{-}{}_{t}^{1}$ bool chan; $t \prec s \vdash$
  (spawn(fn()=>(send(c, true);
                send(d, false)));
   recv(d) andalso recv(c)):bool

Basic Ideas (2): Time Tags

- c: $^{-}{}_{s}^{1}$ bool chan,
  d: $^{-}{}_{t}^{1}$ bool chan; $s \prec t$, $t \prec s$ $\vdash$
  (spawn(fn()=>(send(c, true);
  send(d, false)));
  recv(d) andalso recv(c)):bool

Outline

- Introduction
- Basic Ideas
  - Linear Channels [Kobayashi et al. 96]
  - Time Tags [Kobayashi 97]
- Formalization in process calculi
- Extension by Usage Annotations [Sumii & Kobayashi 98]
- Conclusion

Type Judgment in the Type System

$$G;\ \prec\ \vdash\ \mathbf{P}$$

G : type environment
(mapping from variables to types)

$\prec$: time tag ordering
(binary relation on time tags)

- **P** uses channels according to
  - the usage specified by Γ
  - the order specified by $\prec$

Correctness of the Type System

- Subject Reduction:

"Reduction preserves well-typedness"

$$\Gamma,\ \mathtt{c}{:}\updownarrow_{\mathtt{t}}^{1}\ \mathtt{int\ chan};\ \prec\ \vdash$$
$$\mathtt{(spawn(fn()=>send(c,\ 3));}$$
$$\mathtt{let\ v\ =\ recv(c)\ in}\ \frac{1}{4}\mathtt{)}$$
$$\Downarrow$$
$$\Gamma,\ \mathtt{c}{:}\tfrac{1}{2}{}_{\mathtt{t}}^{1}\ \mathtt{int\ chan};\ \prec\ \vdash$$
$$\mathtt{let\ v\ =\ 3\ in}\ \frac{1}{4}$$

Correctness of the Type System

- Partial Confluence:

"Communication on linear channels won't cause non-determinism"

G; $\prec\ \vdash$ **P** *and* **Q** Ü$_1$ **P** Þ **Q'**

$$\Downarrow$$

Q Þ* **R** *Ü **Q'** *for some* **R**

Correctness of the Type System

- Partial Deadlock-Freedom:

"Non-cyclic communication on linear channels won't cause deadlock"

$$\Gamma; \prec \vdash \mathbf{P}$$

$$\Downarrow$$

$$\mathbf{P} \longrightarrow \mathbf{Q} \textit{ for some } \mathbf{Q}$$

***Unless* P *is trying to receive/send a value from/to some channel typed as* $p_{\mathbf{t}}^{m}$ t chan *where either* $m \neq 1$, $\mathbf{t} \prec^{+} \mathbf{t}$, $p = -$ *or* $\overline{}$**

Outline

- Introduction
- Basic Ideas
  - Linear Channels [Kobayashi et al. 96]
  - Time Tags [Kobayashi 97]
- Formalization in process calculi
- Extension by Usage Annotations [Sumii & Kobayashi 98]
- Conclusion

Generalize Linear Channels by Usage Annotations

U (usage)	::=	**O**	(output)
	|	**I.** U	(input & sequential execution)
	|	U | V	(concurrent execution)
	|	**!** U	(replication)
	|	**-**	(none)

✓ **c:(O|O|I.I.-)$_t$ int chan;** Æ ⊢

```
  (spawn(fn()=>send(c, 3));
   spawn(fn()=>send(c, 7));
   let v = recv(c)
       w = recv(c) in ¼)
```

Generalize Linear Channels by Usage Annotations

Annotate `I`'s and `O`'s with

- Capability (**`c`**)

  "The input/output will succeed (if it is performed)"

- Obligation (**`o`**)

  "The input/output must be performed (though it won't succeed)"

— "Deadlock" if these assumptions don't hold

Generalize Linear Channels by Usage Annotations

"For every* I/O *with capability,
a corresponding* O/I *with obligation"

✓ **c:(O_o|I_c.-)$_t$ int chan; Æ ⊢**
(spawn(fn()=>send(c, 3));
let v = recv(c) in ¼)

✗ **c:(O_o|I_c.-)$_t$ int chan; Æ ⊢**
let v = recv(c) in ¼

Generalize Linear Channels by Usage Annotations

We can uniformly express usage of

- Linear Channels

  $$\mathtt{O}_{\mathtt{co}}|(\mathtt{I}_{\mathtt{co}}.\mathtt{-})$$

- “Semaphore” Channels

  $$\mathtt{O}_{\mathtt{o}}|!(\mathtt{I}_{\mathtt{c}}.\mathtt{O}_{\mathtt{o}})$$

- “Client-Server” Channels

  $$!\mathtt{O}_{\mathtt{c}}|!(\mathtt{I}_{\mathtt{o}}.\mathtt{-})$$

etc.

Usage as LL-Formula

$$[|\ \_\ |] : \mathit{Usage} \to \mathit{LLFormula}$$

$$[|\ \mathtt{O}_{\mathtt{co}}\ |] = \mathbf{m} \qquad [|\ \mathtt{I}_{\mathtt{co}}.\,U\ |] = \mathbf{m} \multimap [|\ U\ |]$$

$$[|\ \mathtt{O}_{\mathtt{c}}\ |] = \mathbf{m} \oplus \mathbf{1} \qquad [|\ \mathtt{I}_{\mathtt{c}}.\,U\ |] = (\mathbf{m} \multimap [|\ U\ |]) \oplus \mathbf{1}$$

$$[|\ \mathtt{O}_{\mathtt{o}}\ |] = \mathbf{m}\ \&\ \mathbf{1} \qquad [|\ \mathtt{I}_{\mathtt{o}}.\,U\ |] = (\mathbf{m} \multimap [|\ U\ |])\ \&\ \mathbf{1}$$

$$[|\ \mathtt{O}\ |] = (\mathbf{m}\ \&\ \mathbf{1}) \oplus \mathbf{1} \qquad [|\ \mathtt{I}.\,U\ |] = ((\mathbf{m} \multimap [|\ U\ |])\ \&\ \mathbf{1}) \oplus$$

$$[|\ U \mid V\ |] = [|\ U\ |] \otimes [|\ V\ |]$$

$$[|\ \text{-}\ |] = \mathbf{1}$$

$$[|\ !U\ |] \qquad !\,[|\ U\ |]$$

Usage as LL-Formula

U is a “reliable” usage

i.e., For every **`I`**/**`O`** *with capability,*

a corresponding **`O`**/**`I`** *with obligation exists*

$$\Updownarrow$$

$[|\ \boldsymbol{U}\ |]$ always reduces to $\mathbf{1}$

i.e., no unexpected garbage (producer/consumer) remains

e.g.

$$[|\ \mathtt{O_o}\,|\,\mathtt{I_c}\,.\,\text{-}\ |] \multimap \mathbf{1} \quad {}^{\circ} \quad (\mathbf{m} \,\&\, \mathbf{1}) \otimes ((\mathbf{m} \multimap \mathbf{1}) \oplus \mathbf{1}) \multimap \mathbf{1}$$

Conclusion

Summary:

"Resource- & order-conscious" type system with *linear channels* & *time tags*

Future Work

- Type Inference Algorithm for Usage Annotations
  (Cf. for time tag ordering [Kobayashi 97],
  for linear channels [Igarashi & Kobayashi 97])
- Aggressive Optimization by the Type Information
- Semantics of Time Tag Ordering
  - Linear Logic with Sequencing Operator?