Session Types

Outline

0. Conception

1. Birth of session types
   (Honda '93)
   (Takeuchi et al. '94)
   (Honda et al. '98)

2. Growing up: Multiparty sessions
   (Honda et al. '08)

3. Mid-life crisis: Canonicity?
   (Daroha et al '17)
   (Caires + Pfenning '10, Wadler '12)

* Meta-themes, discussion to wrap up
* Additional resources + work related to discussion at end

PREAMBLE: RESEARCH CLIMATE

> "SESSION TYPES: TALKS THE TALK BUT DOESN'T WALK THE WALK"
> — GENERIC SKEPTIC

- LOTS OF TALK
  - CITATIONS, HYPE
- ✗ HARD TO FOLLOW THREAD (INCONSISTENT TERMS, IDEAS, NOTATION)
- ✗ SOME BIG MISTAKES

- KINDA WALKS THE WALK
  - NATIVE: SCRIBBLE, RAST, ATS, SEPI, SILL, ETC.
  - LIBRARIES & EXTENSIONS: C, GO, HASKELL, JAVA, PYTHON, RUST
  - DEPLOYED: RED HAT, COGNIZANT, OOI

- GOOD RESEARCH LESSONS (EVEN FOR NON-BELIEVERS)

Preamble: Notation

Goals

1. Self-consistent
2. Consistent with Mitch
3. Consistent with source

- Upshot: usually different than source
- Advice: Read lang. "basics" very carefully
  - Symbol might not mean what you think
  - Subtleties with, esp., binders + scope

↑ = Output/Send, ↓ = Input/Receive

P, Q, R ::=

0	"Nil" process – do nothing
$x\langle v\rangle.P$	Output v on x, then do P
$x(y).P$	Input y from x, then do P
$P \mid Q$	Do P + Q in parallel
$(\nu x)P$	Do P with new channel x
$!P$	Spawn copies of P

STEP 0: CONTEXT

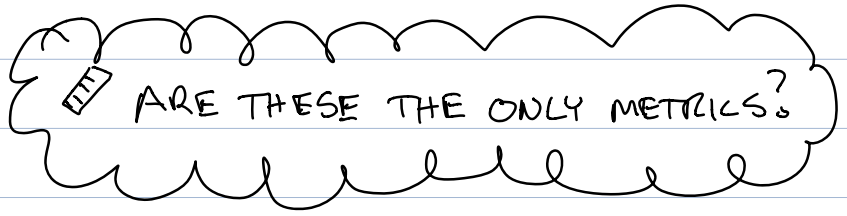
> "... COMPUTER SCIENCE HAS SERIOUSLY TAKEN UP THE CHALLENGE TO UNDERSTAD THE BEHAVIOR OF COMMUNICATING SYSTEMS IN THE SAME WAY AS IT UNDERSTANDS THE BEHAVIOR OF COMPUTER PROGRAMS" - MILNER '99

> "... FOLLOW [THE λ-CALCULUS'] EXAMPLE IN SEEKING SOMETHING SMALL & POWERFUL" - MILNER '93

CAN π-CALCULUS BE AS FUNDAMENTAL & UBIQUITOUS A TOOL AS λ-CALCULUS?

IS IT:

1. "EXPRESSIVE"? ENCODE λ-CALCULUS, OBJECTS
2. ~~IMPLEMENTABLE?~~ PICT, ?
3. EXTENSIBLE? TYPES, EXCEPTIONS, MODULES, ETC



STEP 0 – BEHAVIORS: Ω-LIKE

RECALL: $\Omega \triangleq (\lambda x.x)(\lambda x.x)$

$$
\begin{aligned}
\Omega &= (\lambda x.xx)(\lambda x.xx) && \text{(DEF)} \\
&\to (x\,x)[x \mapsto (\lambda x.xx)] && (\beta) \\
&= (\lambda x.xx)(\lambda x.xx) && \text{(SUBST)} \\
&= \Omega
\end{aligned}
$$

$$
\begin{aligned}
& (\nu\, c)\; !_0(c\langle HB\rangle.\mathbf{0}) \mid !_0(c(x).\mathbf{0}) \\
\equiv\; & (\nu\, c)\; !_0(c\langle HB\rangle.\mathbf{0}) \mid c\langle HB\rangle.\mathbf{0} \mid c(x).\mathbf{0} \mid !_0(c(x).\mathbf{0}) \\
\xrightarrow{c\,\uparrow\,HB}\; & (\nu\, c)\; !_0(c\langle HB\rangle.\mathbf{0}) \mid \mathbf{0} \mid \mathbf{0}[x \mapsto HB] \mid !_0(c(x).\mathbf{0}) \\
\equiv\; & (\nu\, c)\; !_0(c\langle HB\rangle.\mathbf{0}) \mid !_0(c(x).\mathbf{0})
\end{aligned}
$$

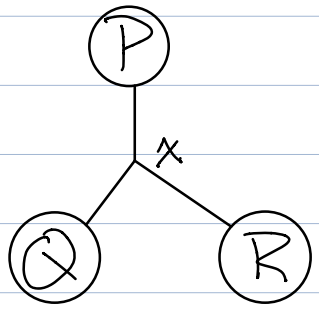
WE WANT THIS (SERVERS!), BUT MAYBE CONTROLLED

BEHAVIORS: INTERFERENCE

$$(\nu x)\ x\langle 5\rangle.P' \mid x(y).Q' \mid x(z).R'$$

$\hookrightarrow^*$ $P \mid Q[y \mapsto 5] \mid R$

OR $P \mid Q \mid R[z \mapsto 5]$



REMEMBER MITCH'S MOLECULAR ACTION EXAMPLE FROM LAST CLASS?

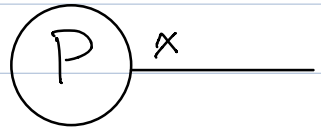
THIS IS USUALLY BAD, UNINTENTIONAL

IDEA: AT MOST TWO PROCESSES PER CHANNEL?

BEHAVIORS: DEADLOCK

$(\nu x)\, x(y).P'$

↳ NEVER STEPS, NON-0



(SO LONELY)

THINK: WHY IS THIS DIFFERENT THAN Ω?

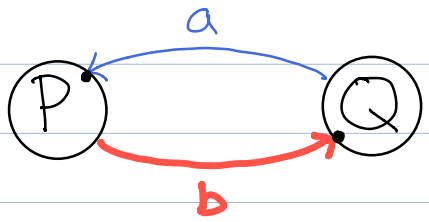
THIS IS DEFINITELY BAD

IDEA: AT LEAST TWO PROCESSES PER CHANNEL?

BEHAVIORS: MORE DEADLOCK

$(\nu ab)\ \ a(x).b\langle x\rangle.P' \mid b(y).a\langle y\rangle.Q'$

↳ NEVER STEPS, NON ①



THESE ARE JUST AS BAD, BUT SNEAKIER

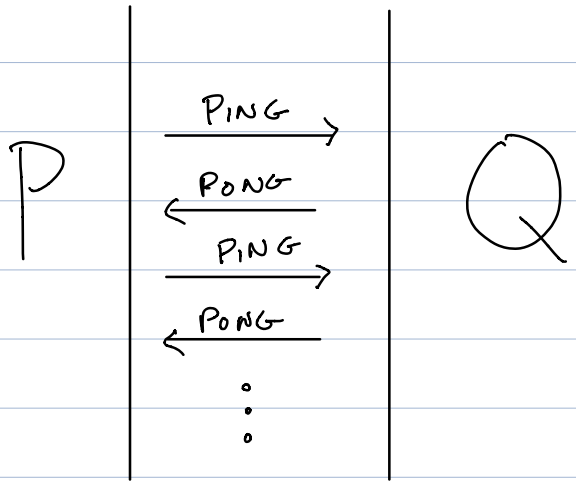
IDEA: ELIMINATE CYCLES?

STEP 0: OTHER TYPE SYSTEMS

1. "... DOES NOT DESCRIBE SEQUENCING OF VALUES COMMUNICATED ALONG A CHANNEL...
2. FOR EXAMPLE, IT CANNOT DESCRIBE ... AN ALTERNATING SEQUENCE...
3. DYNAMIC CONSTRAINTS... DIFFICULT TO EXPRESS WHILE MAINTAINING THE CHARACTERISTICS OF TRADITIONAL TYPE SYSTEMS"

– PIERCE & SANGIORGI '94 (I/O TYPES)

WE THINK OF PROTOCOLS LIKE THIS:



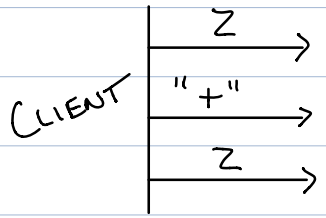
BUT WE HAVE CHANNELS LIKE THIS:

```
c := make(chan Ping)
```

STEP 1a. "TYPES FOR DYADIC INTERACTION" (HONDA '93)

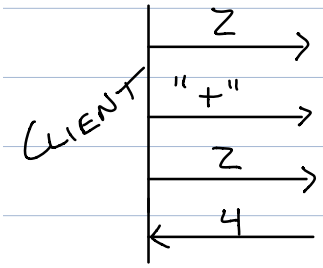
BIG IDEAS

1. A CHANNEL IS USED FOR A SEQUENCE OF ACTIONS



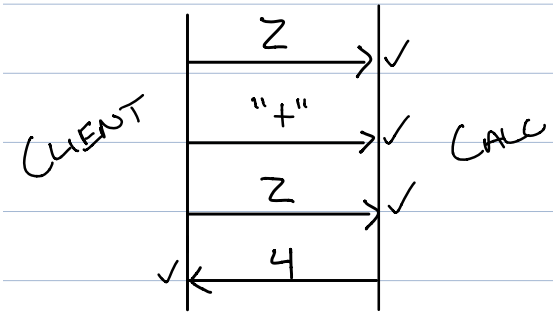
NOT NECESSARILY UNIFORM

2. THE SEQUENCE IS CONVERSATIONAL



FLOWS BOTH WAYS

3. ACTIONS ON EACH END OF A CHANNEL ARE COMPATIBLE



Step 1a: The Action Types

$\uparrow A.B$	Output A, then do B
$\downarrow A.B$	Input A, then do B
$\mathbb{1}$	Do nothing

Ex: Calc : $\downarrow Num.\downarrow Op.\downarrow Num.\uparrow Num.\mathbb{1}$

More:

$A \oplus B$	Choose to do A or B (output)
$A \,\&\, B$	Offer to do A or B (input)

Ex: Smart Calc:

$\&\{$

Neg: $\downarrow Num.\uparrow Num.\mathbb{1}$,

Plus: $\downarrow Num.\downarrow Num.\uparrow Num.\mathbb{1}$,

Divide: $\downarrow Num.\downarrow Num.$

$\oplus\{$

Result: $\uparrow Num.\mathbb{1}$,

DivBy0: $\mathbb{1}$

$\}$

$\}$

Client must offer (&) to handle both cases

Think: What's unrealistic about this "server?"

STEP 1a: COMPATIBILITY

HOW DO WE ENSURE PROCESSES AGREE ON PROTOCOL?

EX: → CALC: $\downarrow$NUM.$\downarrow$OP.$\downarrow$NUM.$\uparrow$NUM.$\mathbb{I}$
→ CLIENT: $\uparrow$NUM.$\uparrow$OP.$\uparrow$NUM.$\downarrow$NUM.$\mathbb{I}$

CAN'T BE ANYTHING ELSE!

IDEA: PROTOCOL FROM ONE POV COMPLETELY DETERMINES THE OTHER

DUALITY

$$\overline{\uparrow A.B} \triangleq \downarrow \bar{A}.\bar{B}$$
$$\overline{\downarrow A.B} \triangleq \uparrow \bar{A}.\bar{B}$$
$$\overline{A \oplus B} \triangleq \bar{A} \,\&\, \bar{B}$$
$$\overline{A \,\&\, B} \triangleq \bar{A} \oplus \bar{B}$$
$$\overline{\mathbb{I}} \triangleq \mathbb{I}$$

"REVERSE THE SIGN + RECUR"

$$\frac{c:A \vdash P \qquad c:\bar{A} \vdash Q}{\vdash (\nu c)\, P \mid Q}$$

Step 1a: Recap

I'm on the hook for...

- Programs??
- Sessions??
- Linearity??
- Type safety??

Flaw: Unrealistic programming model

The new

- Type of channel ≈ protocol in mind
- Duality ⇒ compatability
- Choice (as a primitive)

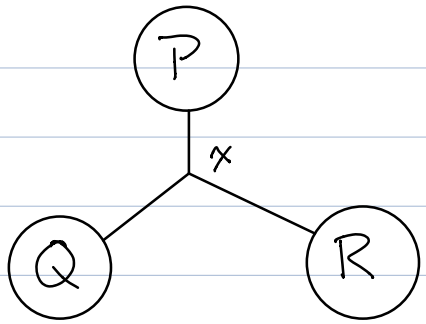
Step 1b: "An Interaction-Based Language and its Typing System" (Takeuchi et al '94)

- Emphasis on Programs + Code Organization
- Yet another π-calc variant (YAPCV)
- How do we use channels?

```
P_CALC(c°. C_ALC) =
    c(left).
    console.log(left).
    c(op ).
    console.log( op ).
    c(right).
    console.log(right).
    case op (
        + ⇒ c<left + right>
        * ⇒ c<left * right>
        _ ⇒ c<0>
    ). 𝟙
```

STEP 1b: IS THIS SAFE?

REMEMBER BEHAVIORS:
> 2 PROCESSES ON CHANNEL ⇒ INTERFERENCE



x@P: ↑INT. ↓BOOL. 𝕀
x@Q: ↓INT. ↑BOOL. 𝕀
x@R: ↓INT. ↑BOOL. 𝕀

∫ x ↑INT

x@P: ↓BOOL. 𝕀
x@Q: ↑BOOL. 𝕀
x@R: ↓INT. ↑BOOL. 𝕀

R WAS LEFT BEHIND!

∫ x ↑BOOL

RUNTIME ERROR: EXPECTED INT, GOT BOOL

IDEA: ENSURE EACH ENDPOINT IS AT 1 PROCESS

Step 16: Linear Channels (Simplified Excerpt)

$\Delta \vdash P$ All channels in Δ must be used – Follow Protocol!

*1. Channel endpoint can't go to two processes

$$\frac{\Delta_1 \vdash P \qquad \Delta_2 \vdash Q}{\Delta_1 + \Delta_2 \vdash P \mid Q}\ \text{Par}$$

2. Channel "takes a step"

$$\frac{\Delta, k : B \vdash P \qquad \vdash v : A}{\Delta, k : \uparrow A.B \vdash k\langle v \rangle.P}\ \text{Output}$$

*3. All channels are used up by the end

$$\frac{}{\varnothing \vdash 0}\ \text{Done}$$

4. Channels must be used in any case

$$\frac{\Delta, k : A \vdash P \qquad \Delta, k : B \vdash Q}{\Delta, k : A \,\&\, B \vdash k.\mathrm{case}(P, Q)}\ \text{Offer}$$

Step 1b: Bootstrapping

- Where do linear channels come from?

- Unrealistic to assume one for all pairs of processes from beginning of time

Remember: Mitch's walkie talkies

$$(\nu s)\,(\ldots \mid (\nu k)\, s\langle k\rangle.P \mid s(k).Q \ldots)$$

Design Pattern ⇒ Primitive

Request: $?s(k:A).P$ (client)

Accept: $!s(k:\bar{A}).Q$ (server)

Public address

Fresh, private, linear **session**

$\Gamma ; \Delta \vdash P$

Unrestricted — Linear

Step 1b. Recap

The New

- Sessions (private, linear) vs. shared names
- Programming faculties to help organize

The old

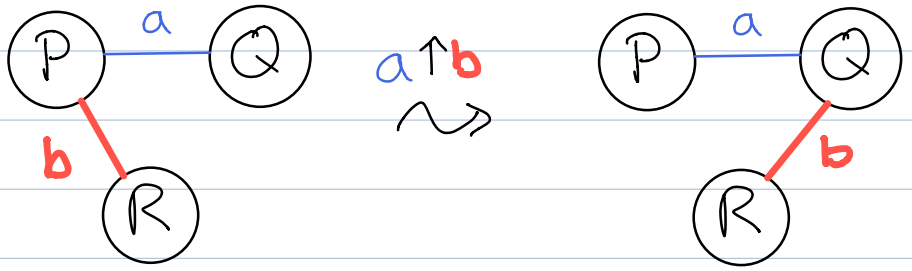
- Action types ⇒ session types
- Duality
- Linearity (but new for you)

Flaws

- No session mobility
- (still) no recursion

STEP 1c: "LANGUAGE PRIMITIVES & TYPE DISCIPLINE FOR STRUCTURED COMMUNICATION-BASED PROGRAMMING" (HONDA ET AL. '98)

- "THE CHOSEN ONE"
- DELEGATION: LIKE SCOPE EXTRUSION, EXCEPT CHANNEL IS MOVED, NOT SHARED



$P = !_0 s(a).\ !_0 s(b).\ a\langle b \rangle.P'$ ← b UNUSABLE

$Q = ?_0 s(a).\ a(b).\ b\langle \text{"HOWDY"} \rangle.Q'$

$R = ?_0 s(b).\ b(x).R'$

↑ "HOWDY" FROM Q

RECOVER MOBILITY W/O HURTING LINEARITY

- RECURSION: PROTOCOLS CAN LOOP

  PINGPONG: $\mu LOOP.\uparrow PING.\downarrow PONG.LOOP$

Step 1: ATM Example

ATM

1. Validate PIN
2. Offer Deposit, Withdraw, Help, or Quit
3. Multiple transactions per visit
4. Help is delegated to customer service

T_{ATM} =

↓PIN.⊕{
  OK: M MENU.
    &{
      DEPOSIT: ↓NAT. MENU,
      WITHDRAW: ↓NAT.↑NAT. MENU,
      HELP: M CHAT.
        ↑STR.&{
          TYPING: ↓STR. CHAT,
          QUIT: 𝟙
        },
      QUIT: 𝟙
    },
  ERR: 𝟙
}

Might delegate remainder of session to customer service

STEP 1: EVERYTHING SO FAR

THE NEW:

- SESSION TYPES ≈ PROTOCOL
- DUALITY ENSURES COMPATIBILITY
- SYNCHRONIZE ON SHARED NAMES TO ESTABLISH SESSIONS
- CHOICE IS HELPFUL

THE OLD (PRE SESSION TYPES):

- LINEARITY
- MOBILITY
- RECURSION

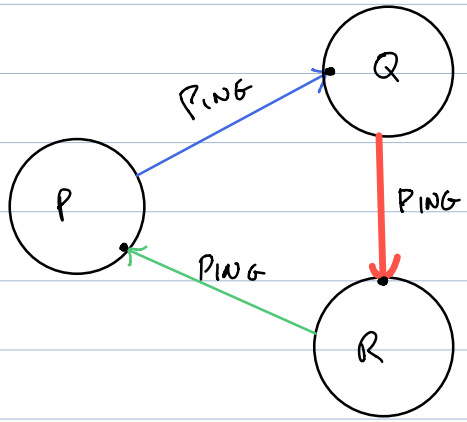
BEHAVIORS

- BASIC TYPE CHECK ✓
- ⅄ IS POSSIBLE (SERVERS!) ✓
- NO INTERFERENCE ON SESSIONS ✓
- DEADLOCK ?

STEP 2: "MULTIPARTY ASYNCHRONOUS SESSION TYPES" (HONDA ET AL. '08)

WHAT ABOUT DEADLOCKS?

$P = c(x).a\langle x\rangle.0$
$Q = a(y).b\langle y\rangle.0$
$R = b(z).c\langle z\rangle.0$



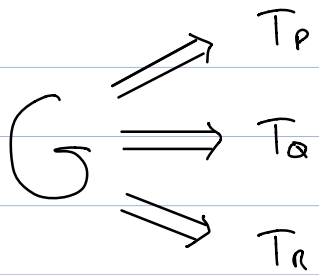
PROBLEM: SESSIONS ARE COMPLETELY INDEPENDENT

IDEA: TAKE GLOBAL VIEW THAT RELATES THEM

REMEMBER: WITH JUST ONE SESSION, ONE ENDPOINT DERIVABLE FROM OTHER

$A \Rightarrow \bar{A}$

IDEA: DERIVE ENDPOINTS FROM GLOBAL VIEW



Step 2: A Tour of MPST

Global Types: Specify Protocol with ALL Parties

$$
\begin{aligned}
G = {} & P \to Q : a\langle \mathrm{Str} \rangle ; \\
& Q \to R : b\langle \mathrm{Str} \rangle ; \\
& R \to P : c\langle \mathrm{Str} \rangle ; \\
& \mathbb{1}
\end{aligned}
$$

Projection to Local Types: Derive Protocol from POV of Each Party

$$
\begin{aligned}
G \upharpoonright P &= a \uparrow \mathrm{Str} ; c \downarrow \mathrm{Str} ; \mathbb{1} \\
G \upharpoonright Q &= a \downarrow \mathrm{Str} ; b \uparrow \mathrm{Str} ; \mathbb{1} \\
G \upharpoonright R &= b \downarrow \mathrm{Str} ; c \uparrow \mathrm{Str} ; \mathbb{1}
\end{aligned}
$$

Local Type vs. BST

$a \uparrow \mathrm{Str} ; c \downarrow \mathrm{Str} ; \mathbb{1}$ — Captures interleaving

vs.

$a : \uparrow \mathrm{Str} ; \mathbb{1}$

$c : \downarrow \mathrm{Str} ; \mathbb{1}$ — YMMV; deadlock if used in wrong order

STEP 2: ATM REVISITED

```
GATM =
  CLIENT → ATM : user{
    DEPOSIT : ..., WITHDRAW : ...,
    QUIT :
      ATM → CSERV : backend{ QUIT : 1 },
    HELP :
      ATM → CSERV : backend{
        HELP : ATM → CSERV : backend<
          user↓STR.user↑STR
        >
      }
  }
```

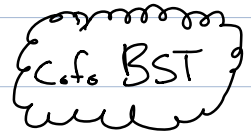
DELEGATION

```
GATM↑CSERV =
  backend &{
    QUIT : 1,
    HELP :
      backend↓<user↓STR.user↑STR>
  }
```

STEP 2° RECAP

THE NEW

- GLOBAL TYPES
- PROJECTION ⇒ COMPATABILITY
- LOCAL TYPES WITH SESSION INTERLEAVING
- DEADLOCK GUARANTEES



THE OLD

- LINEARITY
- INTERACTION OPERATORS ($\uparrow, \downarrow, \&, \oplus$)
- DUAL ⇒ PROJECTION

STEP 3: IS IT RIGHT?

- ✓ IMPLEMENTATIONS!
- ✓ EXTENSIONS!
- ? "EXPRESSIVE"
- ✗ BUGS
- ✗ FRAGMENTATION

CAN WE:

a) SIMPLIFY IT?

b) MAKE IT MORE FUNDAMENTAL?

GOAL: CANONICITY

STEP 3a: SIMPLIFY

EX: "SESSION TYPES REVISITED" (DARDHA ET AL. '17)

IDEA: EMBED SESSION TYPES IN KPT96 LINEAR I/O (BINARY)

RECALL LINEAR I/O:

$\uparrow^1$ Num	CHANNEL OUTPUTS NUM EXACTLY ONCE
$\downarrow^1$ Num	" INPUTS "
$\updownarrow^1$ Num	" OUTPUTS + INPUTS "

- REPLACE 1 WITH ω FOR UNRESTRICTED
- CAN SEND CHANNELS OVER CHANNELS

BSI: SESSION USED MULTIPLE TIMES, BUT EXACTLY ONCE AT EACH STEP

VS.

LINEAR I/O: CHANNEL USED EXACTLY ONCE (FOR SENDING, RECEIVING)

Step 3a: CPS Encoding

- Ex: $[\![\uparrow \text{PING} . \downarrow \text{PONG} . \mathbb{1}]\!]$

$$= \uparrow^1 (\text{PING}, \underbrace{\uparrow^1 \text{PONG}})$$

"continuation" channel for partner to respond with

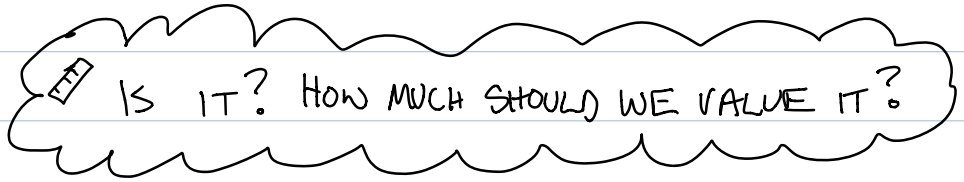
- More generally,

$$[\![\uparrow A . B]\!] = \uparrow^1 ([\![A]\!], [\![B]\!]) \quad \text{output}$$

$$[\![\downarrow A . B]\!] = \downarrow^1 ([\![A]\!], [\![B]\!]) \quad \text{input}$$

STEP 3b: "MORE FUNDAMENTAL"

- HISTORY OF APPEALING TO LOGIC FOR CS IDEAS (OR TO VALIDATE EXISTING CS IDEAS)
- "DOUBLE DISCOVERY" IS A GOOD SIGN!

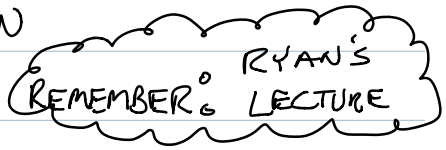


- EXAMPLES:

STLC ≈ NATURAL DEDUCTION

POLYMORPHIC λ ≈ SYSTEM F

CALL/CC ≈ PIERCE'S LAW



IS THERE A LOGICAL FOUNDATION FOR π-CALCULUS?

Background: Curry-Howard Correspondence

Look Familiar?

$$\frac{\Gamma \vdash A \to B \quad \Gamma \vdash A}{\Gamma \vdash B} \to \text{Elim}$$

Logic	STLC
Proposition	Type
Proof	Program
Proof Simplification	Computation
Cut Elimination	Termination
Implication	Function Type
Conjunction	Product
Disjunction	Sum
Intro Rule	Constructor
Elim Rule	App, Pattern Match

Step 3b: Linear Logic ≈ STPC

"Intuitionistic Linear Propositions as Session Types"
(Caires → Pfenning '10)

Linear Logic	Session-Typed π-Calculus (STPC)
Proposition	Session Type
Proof	Process
Proof Simplification	Communication
Cut Elimination	Deadlock-Freedom
Mult Connectives	Select / Offer
Add Connectives	Output / Input
Exponentials	Request/Accept
Identity Axiom	Forwarding
Cut (Entailment)	Parallel Composition

Goal: Use **Logic** to guide development of Rules

REFLECTIONS

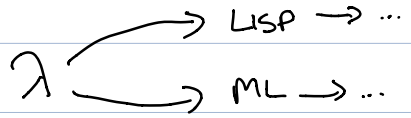
THEMES:

- HOW DO WE EVALUATE A THEORY? HOW SHOULD WE?
- HOW DOES THE EVOLUTION OF THE π-CALCULUS MIRROR THAT OF THE λ-CALCULUS?

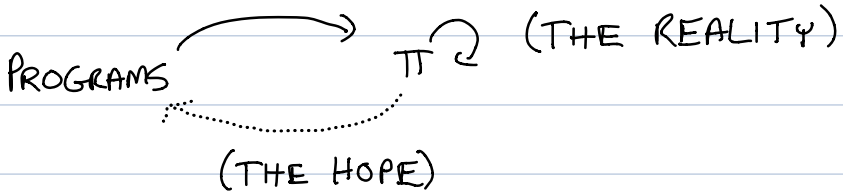
	λ	π
FOUNDED	20s-30s	1992
ROOTS	LOGIC, MATH	PROGRAMS, PROTOCOLS
LOGIC LINK	'20s, '50s, '60s	2010
↳PROP	TYPE	SESSION
↳PROOF	PROGRAM	PROCESS
↳PROOF RED.	EVALUATION	COMMUNICATION
↳GUARANTEE	TERMINATION	DEADLOCK-FREEDOM
GENERALITY	M	!
IMPLS	LISP('53), ML('73), ETC.	PICT ('92-'98, RIP)

Reflections: Evolution

- Functional PLs were built with λ as inspiration

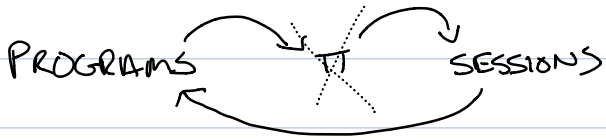


- ππ was based on real programs



  - Way more inertia (~30-40 yrs, $$)
  - Fool's errand?

- Session types



- Spirit of ππ (Honda's "Joyful Hacking")
- Don't require totally new languages

! Thought about human in the loop

Resources

- For main papers, see abstract on website
- "Session-Typed Concurrent Programming" on YouTube
  - OPLSS '19 w/ Frank Pfenning
  - OPLSS '18 w/ Stephanie Balzer
- Nobuko Yoshida's talk @ Papers We Love
- Marco Carbone's talk @ FrIDA 2020
- "An Introduction to Session Types" on Wen Kokke's blog
- "Fundamentals of Session Types" – Vasco T. Vasconcelos, '12
- "Foundations of Session Types & Behavioral Contracts" – Hüttel et al, 2016
- Honda's "Idioms for Interaction" lecture notes

From Discussion

- Relevant to Matthias' 100-Drone example: "Dynamic Multirole Sessions Types" – Denielou & Yoshida '11
- On classical vs. constructive logic: Caires & Pfenning '10 use constructive, while Wadler '12 uses classical