Ergometric and Temporal Session Types

Frank Pfenning
Carnegie Mellon University
Joint work with Ankush Das and Jan Hoffmann

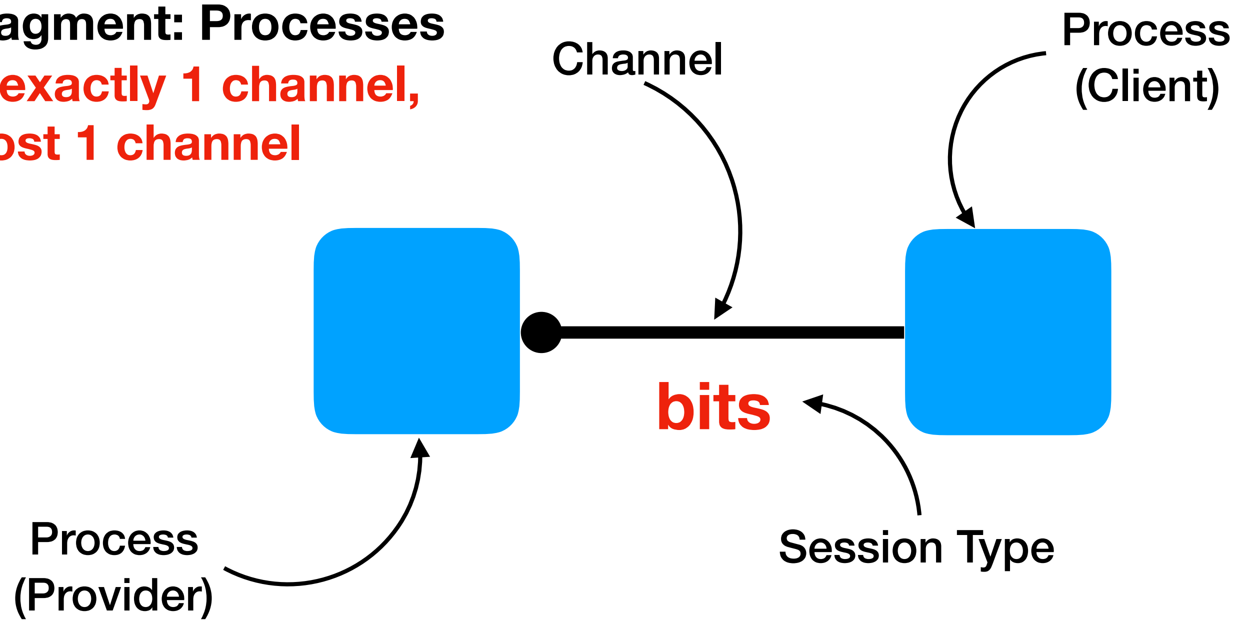
Outline

- Part I: Session Types (and Session-Typed Programs)
- Part II: Capturing Work: Ergometric Types
- Part III: Capturing Time: Temporal Types

Part I
What is a session type?
And a session-typed program?

Bit Streams

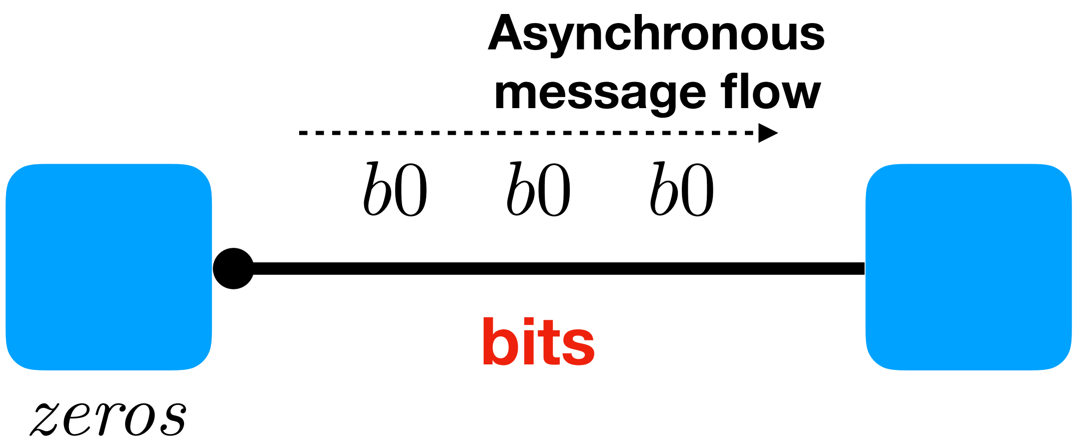
For this talk, for simplicity we restrict ourselves to the subsingleton fragment: Processes provide along exactly 1 channel, use at most 1 channel



$$\mathsf{bits} = \oplus\{b0 : \mathsf{bits}, b1 : \mathsf{bits}\}$$

Internal Choice (provider sends)

Bit Streams



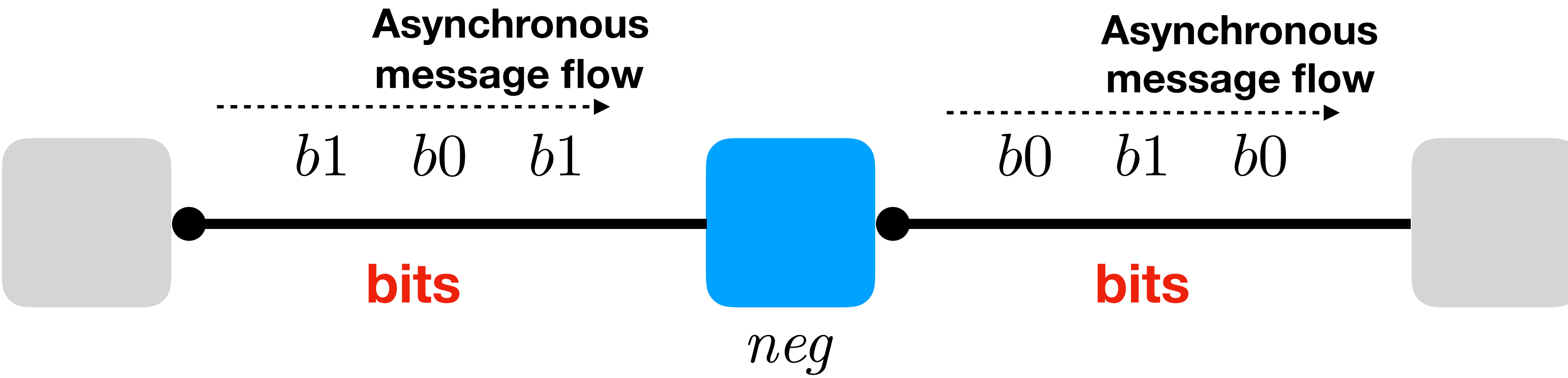
$$\mathsf{bits} = \oplus\{b0 : \mathsf{bits}, b1 : \mathsf{bits}\}$$

$$\vdash zeros : \mathsf{bits}$$

$$zeros = \mathsf{R}.b0\ ;\ zeros$$

R ("right"): send from provider to client

Bit Stream Transducer



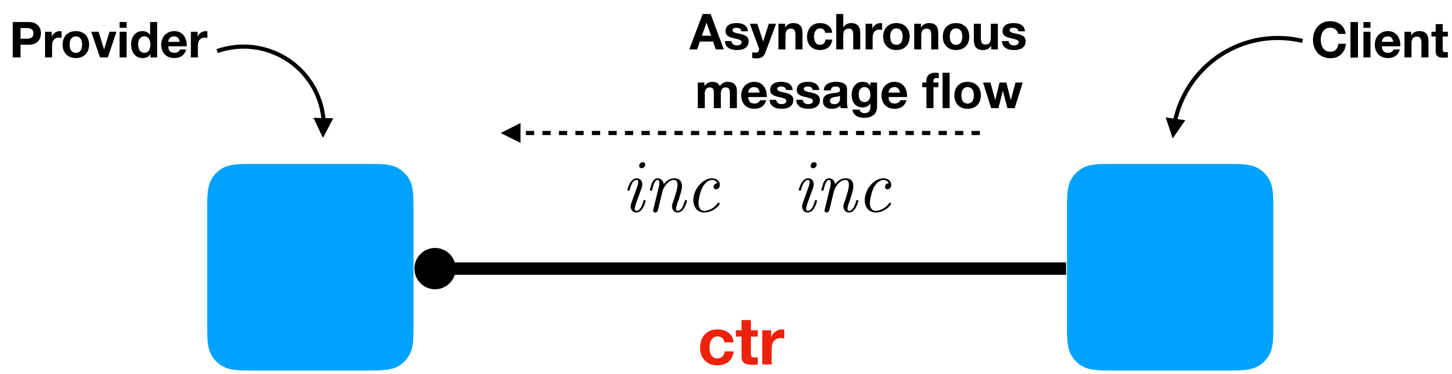
$$\mathsf{bits} = \oplus\{b0 : \mathsf{bits}, b1 : \mathsf{bits}\}$$

$$\mathsf{bits} \vdash neg : \mathsf{bits}$$

$$\begin{aligned} neg = \mathsf{caseL}\ (\, & b0 \Rightarrow \mathsf{R}.b1 \ ;\ neg \\ \mid\ & b1 \Rightarrow \mathsf{R}.b0 \ ;\ neg\,) \end{aligned}$$

caseL: receive from “left” (provider)

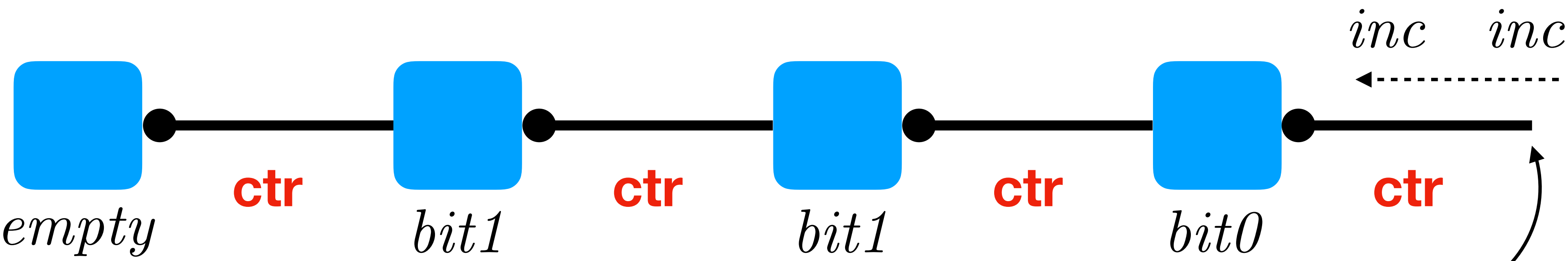
Counter



$$\mathsf{ctr} = \&\{inc : \mathsf{ctr}\}$$

External Choice (client sends)

Binary Counter



Represents number 6 = $(110)_2$

$$\mathsf{ctr} = \&\{inc : \mathsf{ctr}\}$$

$$\vdash empty : \mathsf{ctr}$$
$$\mathsf{ctr} \vdash bit0 : \mathsf{ctr}$$
$$\mathsf{ctr} \vdash bit1 : \mathsf{ctr}$$

Parallel Composition
(spawning a new process)
(associative, but not commutative here!)

$$bit0 = \mathsf{caseR}\,(\, inc \Rightarrow bit1\,)$$
$$bit1 = \mathsf{caseR}\,(\, inc \Rightarrow \mathsf{L}.inc\ ;\ bit0\,)$$
$$empty = \mathsf{caseR}\,(\, inc \Rightarrow empty \,||\, bit1\,)$$

Session Types So Far

$$
\begin{array}{llll}
A, B & ::= & \oplus\{\ell : A_\ell\}_{\ell \in L} & \text{(internal choice)} \\
 & \mid & \&\{\ell : A_\ell\}_{\ell \in L} & \text{(external choice)} \\
 & \mid & a & \text{(def)}
\end{array}
$$

$$
\begin{array}{llll}
P, Q & ::= & \mathsf{R}.k \;;\; P \;\mid\; \mathsf{caseL}\,(\ell \Rightarrow Q_\ell)_{\ell \in L} & (\oplus) \\
 & \mid & \mathsf{caseR}\,(\ell \Rightarrow P_\ell)_{\ell \in L} \;\mid\; \mathsf{L}.k \;;\; Q & (\&) \\
 & \mid & P \parallel Q & \text{(spawn)} \\
 & \mid & f & \text{(def)}
\end{array}
$$

$A \vdash P : B$ (typing judgment)

$P \longrightarrow Q$ (transition judgment)

Typing and Reduction

$$\frac{(\forall \ell \in L) \quad A \vdash P_\ell : B_\ell}{A \vdash \mathsf{caseR}\ (\ell \Rightarrow P_\ell)_{\ell \in L} : \&\{\ell : B_\ell\}_{\ell \in L}} \ \&R$$

$$\frac{(k \in L) \quad B_k \vdash Q : C}{\&\{\ell : B_\ell\}_{\ell \in L} \vdash (\mathsf{L}.k\ ;\ Q) : C} \ \&L$$

$$\frac{A \vdash P : B \quad B \vdash Q : C}{A \vdash (P \mid\mid Q) : C} \ \mathsf{cut}$$

$$\begin{array}{lcll} \mathsf{caseR}\ (\ell \Rightarrow P_\ell) \mid\mid (\mathsf{L}.k\ ;\ Q) & \longrightarrow & P_k \mid\mid Q & (\&C) \\ (\mathsf{R}.k\ ;\ P) \mid\mid \mathsf{caseL}\ (\ell \Rightarrow Q_\ell) & \longrightarrow & P \mid\mid Q_k & (\oplus C) \end{array}$$

Identity as Forward

$$\frac{}{A \vdash \leftrightarrow : A}\ \text{id}$$

Logically: identity
Operationally: forwarding

$$P \parallel (\leftrightarrow) \parallel Q \quad \longrightarrow \quad P \parallel Q$$

Connection to Linear Logic

- Curry-Howard correspondence to (additive) linear logic, plus recursive types and recursive processes
- Extension to (intuitionistic) linear logic adds termination (1), channel receive ($A \multimap B$), channel send ($A \otimes B$), replication ($!A$)
- Programs derived from sequent calculus proofs
- Operational semantics derived from cut reduction
- Satisfies the usual preservation and progress properties
- Synchronous and asynchronous communication interdefinable

Summary

$\oplus\{\ell : A_\ell\}_{\ell \in L}$	send $k \in L$	cont as A_k
$\&\{\ell : A_\ell\}_{\ell \in L}$	recv $k \in L$	cont as A_k
$\mathbf{1}$	send end	(terminate)
$A \multimap B$	recv channel $c : A$	cont as B
$A \otimes B$	send channel $c : A$	cont as B

Generalized Typing Judgment

$$(c_1 : A_1) \ldots (c_n : A_n) \vdash P :: (c : B)$$

Channels Used **Channel Provided**

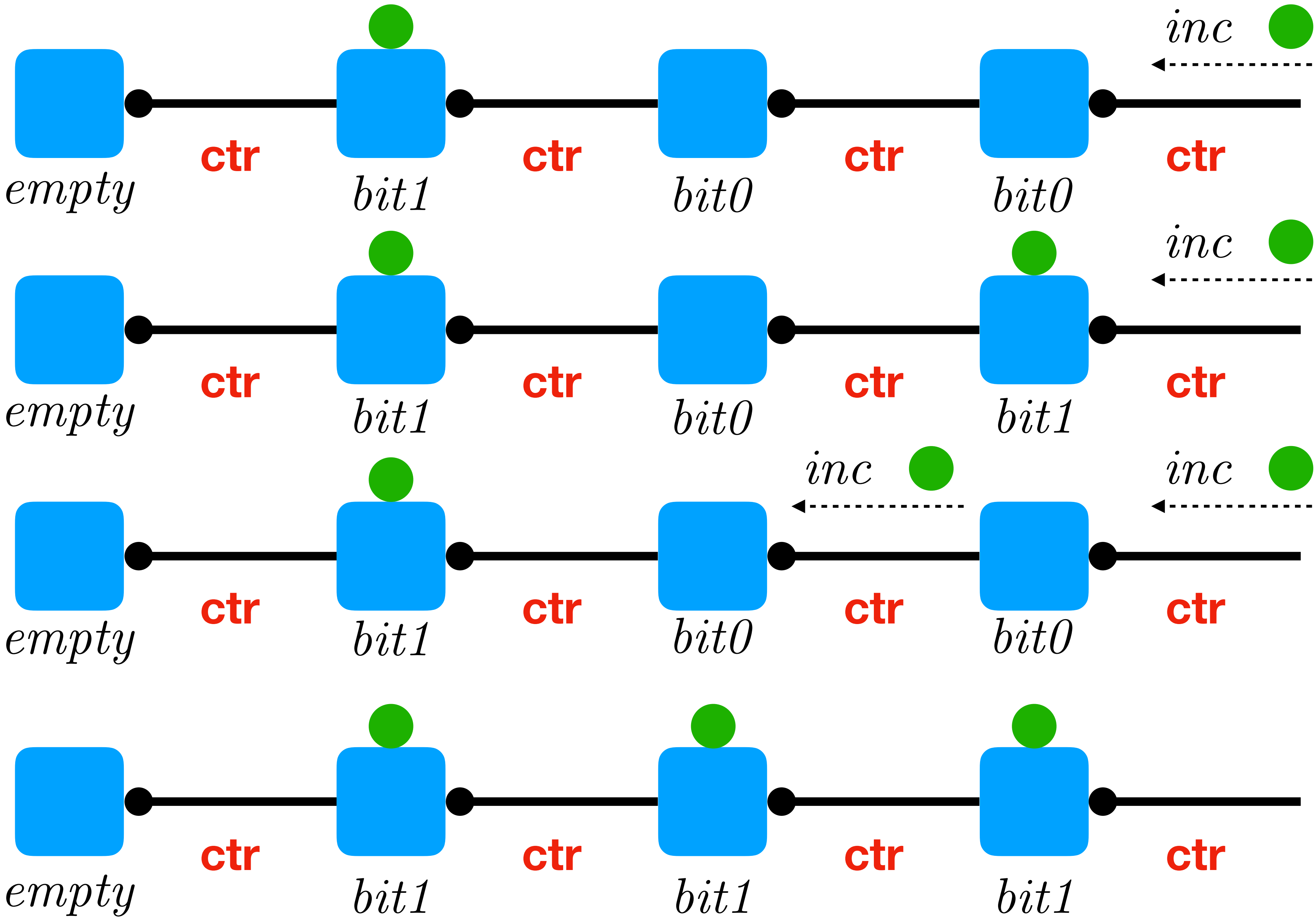
Part II
Capturing Work

Ergometric Types

- Design goals
  - Flexible: support different cost models, e.g., messages sent or processes spawned
  - Conservative: extend rather than redefine type system or logic
  - Compositional: describe individual processes, not just whole programs
  - Precise: capture work accurately
  - General: allow many algorithms and programs to be analyzed
  - Intuitive: predictable, with good error messages
  - Automatic: infer bounds where possible

Amortized Analysis

- We can store tokens (permitting work) with a process
- We can transfer tokens between processes
- Cost model: every send action costs 1 token
- Classic example: binary counter
  - Every bit1 process stores 1 token so it can send a carry bit
  - Every increment message carries 1 token, to be stored
  - When starting from zero, incrementing *n* times requires work *2n*



inc
empty
ctr
bit1
ctr
bit0
ctr
bit0
ctr
inc
empty
ctr
bit1
ctr
bit0
ctr
bit1
ctr
inc
inc
empty
ctr
bit1
ctr
bit0
ctr
bit0
ctr
empty
ctr
bit1
ctr
bit1
ctr
bit1
ctr

Ergometric types in three steps

Step 1
Generalize the Judgment

Judging Potential

Potential r ≥ 0

$$A \vdash^{r} P : C$$

$$\frac{A \vdash^{r} P : C}{A \vdash^{r+1} \text{work} \; ; P : C}\ \text{work}$$

$$\frac{A \vdash^{r} P : B \quad B \vdash^{s} Q : C}{A \vdash^{r+s} (P \mid\mid Q) : C}\ \text{cut} \qquad \frac{}{A \vdash^{0} \leftrightarrow : A}\ \text{id}$$

Parametric Right/Left Rules

$$\frac{(\forall \ell \in L) \quad A \vdash^{r} Q_\ell : B_\ell}{A \vdash^{r} \mathsf{caseR}\ (\ell \Rightarrow P_\ell)_{\ell \in L} : \&\{\ell : B_\ell\}_{\ell \in L}} \ \&R$$

$$\frac{(k \in L) \quad B_k \vdash^{r} Q : C}{\&\{\ell : B_\ell\}_{\ell \in L} \vdash^{r} \mathsf{L}.k\ ;\ Q : C} \ \&L$$

Send, receive, spawn, forward are all free!
For now…

Step 2
Internalize Potential
in Types

Transferring Potential

$$A ::= \ldots \mid \triangleleft^{s} A \mid \triangleright^{s} A$$

Symmetric transfer from provider to client

Transfer of potential manifest in the type!

Transfer of potential s from client to provider

$$\frac{A \vdash^{r+s} P : B}{A \vdash^{r} (\mathsf{getR}^{s} \; ; P) : \triangleleft^{s} B} \triangleleft R \qquad \frac{B \vdash^{t} Q : C}{\triangleleft^{s} B \vdash^{s+t} (\mathsf{payL}^{s} \; ; Q) : C} \triangleleft L$$

$$(\mathsf{getR}^{s} \; ; P)^{r} \quad \| \quad (\mathsf{payL}^{s} \; ; Q)^{s+t}$$

$$\longrightarrow \qquad (P)^{r+s} \quad \| \quad (Q)^{t}$$

Potential of a running process

Step 3
Define Cost Model

Send Actions Cost 1 Token

$$
\begin{array}{lcll}
(P \mid\mid Q)^* & = & (P)^* \mid\mid (Q)^* & \text{cut} \\
(\leftrightarrow)^* & = & \leftrightarrow & \text{id} \\
(\text{caseR}\,(\ell \Rightarrow P_\ell)_{\ell \in L})^* & = & \text{caseR}\,(\ell \Rightarrow (P_\ell)^*)_{\ell \in L} & (\&R) \\
(\text{L}.k \;;\; Q)^* & = & \color{blue}{\text{work}} \;;\; \text{L}.k \;;\; (Q)^* & (\&L) \\
(\text{R}.k \;;\; P)^* & = & \color{blue}{\text{work}} \;;\; \text{R}.k \;;\; (P)^* & (\oplus R) \\
(\text{caseL}\,(\ell \Rightarrow Q_\ell)_{\ell \in L})^* & = & \text{caseL}\,(\ell \Rightarrow (Q_\ell)^*)_{\ell \in L} & (\oplus L)
\end{array}
$$

Blue work is inserted to reflect cost model, not by programmer

Binary Counter Revisited

$$\mathsf{ctr} = \&\{ inc : \triangleleft^{1}\ \mathsf{ctr} \}$$

$$\mathsf{ctr} \vdash^{0} \ bit0 : \mathsf{ctr}$$

$$\mathsf{ctr} \vdash^{1} \ bit1 : \mathsf{ctr}$$

$$\vdash^{0} \ empty : \mathsf{ctr}$$

$$bit0 = \overset{(0)}{\mathsf{caseR}}\,(\, inc \Rightarrow \mathsf{getR}^{1} \overset{(1)}{;}\ bit1\,)$$

$$bit1 = \overset{(1)}{\mathsf{caseR}}\,(\, inc \Rightarrow \mathsf{getR}^{1} \overset{(2)}{;}\ \mathsf{work} \overset{(1)}{;}\ \mathsf{L}.inc \overset{(1)}{;}\ \mathsf{payL}^{1} \overset{(0)}{;}\ bit0\,)$$

$$empty = \overset{(0)}{\mathsf{caseR}}\,(\, inc \Rightarrow \mathsf{getR}^{1} \overset{(1)}{;}\ empty \overset{(0)}{||}\ \overset{(1)}{bit1}\,)$$

Typing Closed Programs

$$\mathsf{ctr} \vdash^{\color{red}{6}} \mathit{plus3} : \mathsf{ctr}$$

$$\begin{aligned}\mathit{plus3} = {} & \overset{6}{\color{blue}{\mathsf{work}}} \overset{5}{;} \mathsf{R}.\mathit{inc} \overset{5}{;} \mathsf{payL} \ ; \\ & \overset{4}{\color{blue}{\mathsf{work}}} \overset{3}{;} \mathsf{R}.\mathit{inc} \overset{3}{;} \mathsf{payL} \ ; \\ & \overset{2}{\color{blue}{\mathsf{work}}} \overset{1}{;} \mathsf{R}.\mathit{inc} \overset{1}{;} \mathsf{payL} \ ; \\ & \overset{0}{\leftrightarrow}\end{aligned}$$

$$\vdash^{\color{red}{12}} \mathit{six} : \mathsf{ctr}$$

$$\mathit{six} = \overset{0}{\mathit{empty}} \ || \ \overset{6}{\mathit{plus3}} \ || \ \overset{6}{\mathit{plus3}}$$

Ergometric Types

- ☑ Flexible: support different cost models, e.g., messages sent or processes spawned
- ☑ Conservative: extend rather than redefine type system or logic
- ☑ Compositional: describe individual processes, not just whole progs.
- ☑ Precise: capture work accurately
- ■ General: allow many algorithms and programs to be analyzed
- ☑ Intuitive: predictable, with good error messages
- ■ Automatic: infer bounds where possible

Work Reconstruction

$$\mathsf{ctr} = \&\{inc : \triangleleft^{1}\ \mathsf{ctr}\}$$

$$\vdash^{0}\ empty : \mathsf{ctr}$$

$$\mathsf{ctr} \vdash^{0}\ bit0 : \mathsf{ctr}$$

$$\mathsf{ctr} \vdash^{1}\ bit1 : \mathsf{ctr}$$

$$bit0 = \mathsf{caseR}\,(\ inc \Rightarrow\ bit1\,)$$
$$bit1 = \mathsf{caseR}\,(\ inc \Rightarrow \mathsf{L}.inc\ ;\ bit0\,)$$
$$empty = \mathsf{caseR}\,(\ inc \Rightarrow\ empty \mid\mid bit1\,)$$

$$\mathsf{ctr} \vdash^{6}\ plus3 : \mathsf{ctr}$$
$$plus3 = \mathsf{R}.inc\ ;\ \mathsf{R}.inc\ ;\ \mathsf{R}.inc\ ;\ \leftrightarrow$$

Reading the Counter Value

$$\mathsf{bits} = \oplus\{b0 : \mathsf{bits}, b1 : \mathsf{bits}, \$: \mathbf{1}\}$$
$$\mathsf{ctr} = \&\{inc : \mathsf{ctr}, val : \mathsf{bits}\}$$

Terminating and closing channel

$$\begin{aligned} \mathit{bit0} = \mathsf{caseR}\ (\ & inc \Rightarrow \mathit{bit1} \\ \mid\ & val \Rightarrow \mathsf{R}.b0\ ;\ \mathsf{L}.val\ ;\ \leftrightarrow\ ) \end{aligned}$$

$$\begin{aligned} \mathit{bit1} = \mathsf{caseR}\ (\ & inc \Rightarrow \mathsf{L}.inc\ ;\ \mathit{bit1} \\ \mid\ & val \Rightarrow \mathsf{R}.b1\ ;\ \mathsf{L}.val\ ;\ \leftrightarrow\ ) \end{aligned}$$

$$\begin{aligned} \mathit{empty} = \mathsf{caseR}\ (\ & inc \Rightarrow \mathit{empty} \mid\mid \mathit{bit1} \\ \mid\ & val \Rightarrow \mathsf{R}.\$\ ;\ \mathsf{closeR}\ ) \end{aligned}$$

Parametric Bounds

$$\mathsf{bits} = \oplus\{b0 : \mathsf{bits}, b1 : \mathsf{bits}, \$: \mathbf{1}\}$$

$$\mathsf{ctr} = \&\{inc : \mathsf{ctr}, val : \mathsf{bits}\}$$

Every bit0 and bit1 process stores an additional 2 tokens

$$\mathsf{ctr} = \&\{inc : \triangleleft^{3}\, \mathsf{ctr}, val : \triangleleft^{2}\, \mathsf{bits}\}$$

$$\mathsf{ctr}[n] = \&\{inc : \triangleleft^{1}\, \mathsf{ctr}[n+1], \quad val : \triangleleft^{2\lceil \log(n+1) \rceil + 2}\, \mathsf{bits}\,\}$$

“Internal measure” to express parametric bound

Client provides enough tokens at the end to read out value

Other Examples

$$\mathsf{stack}_A = \&\{\ \mathsf{ins} : A \multimap \mathsf{stack}_A,\ \mathsf{del} : \triangleleft^2 \oplus\{\ \mathsf{none} : \mathbf{1},\ \mathsf{some} : A \otimes \mathsf{stack}_A\ \}\ \}$$

$$\mathsf{queue'}_A = \&\{\ \mathsf{ins} : \triangleleft^6 (A \multimap \mathsf{queue'}_A),\ \mathsf{del} : \triangleleft^2 \oplus\{\ \mathsf{none} : \mathbf{1},\ \mathsf{some} : A \otimes \mathsf{queue'}_A\ \}\ \}$$

Queue as two stacks

$$\mathsf{queue}_A[n] = \&\{\ \mathsf{ins} : \triangleleft^{2n} (A \multimap \mathsf{queue}_A[n+1]),\ \mathsf{del} : \triangleleft^2 \oplus\{\ \mathsf{none} : \exists\{n = 0\}\, \mathbf{1},\ \mathsf{some} : \exists\{n > 0\}\, A \otimes \mathsf{queue}_A[n-1]\ \}\ \}$$

Queue as bucket brigade

More Examples

$$\mathsf{list}_A^r = \oplus\{\ \mathsf{nil} : \mathbf{1},\ \mathsf{cons} : \triangleright^r\,(A \otimes \mathsf{list}_A^r)\ \}$$

$$(l_1 : \mathsf{list}_A^2)\,(l_2 : \mathsf{list}_A^0) \vdash^0 \ \mathit{append} :: (l : \mathsf{list}_A^0)$$

$$\mathsf{mapper}_{AB} = \&\{\ \mathsf{next} : A \multimap (B \otimes \triangleright^2\,\mathsf{mapper}_{AB}),\ \mathsf{done} : \mathbf{1}\ \}$$

$$(l : \mathsf{list}_A^2)\,(m : \mathsf{mapper}_{AB}) \vdash^2 \ \mathit{map} :: (k : \mathsf{list}_B^0)$$

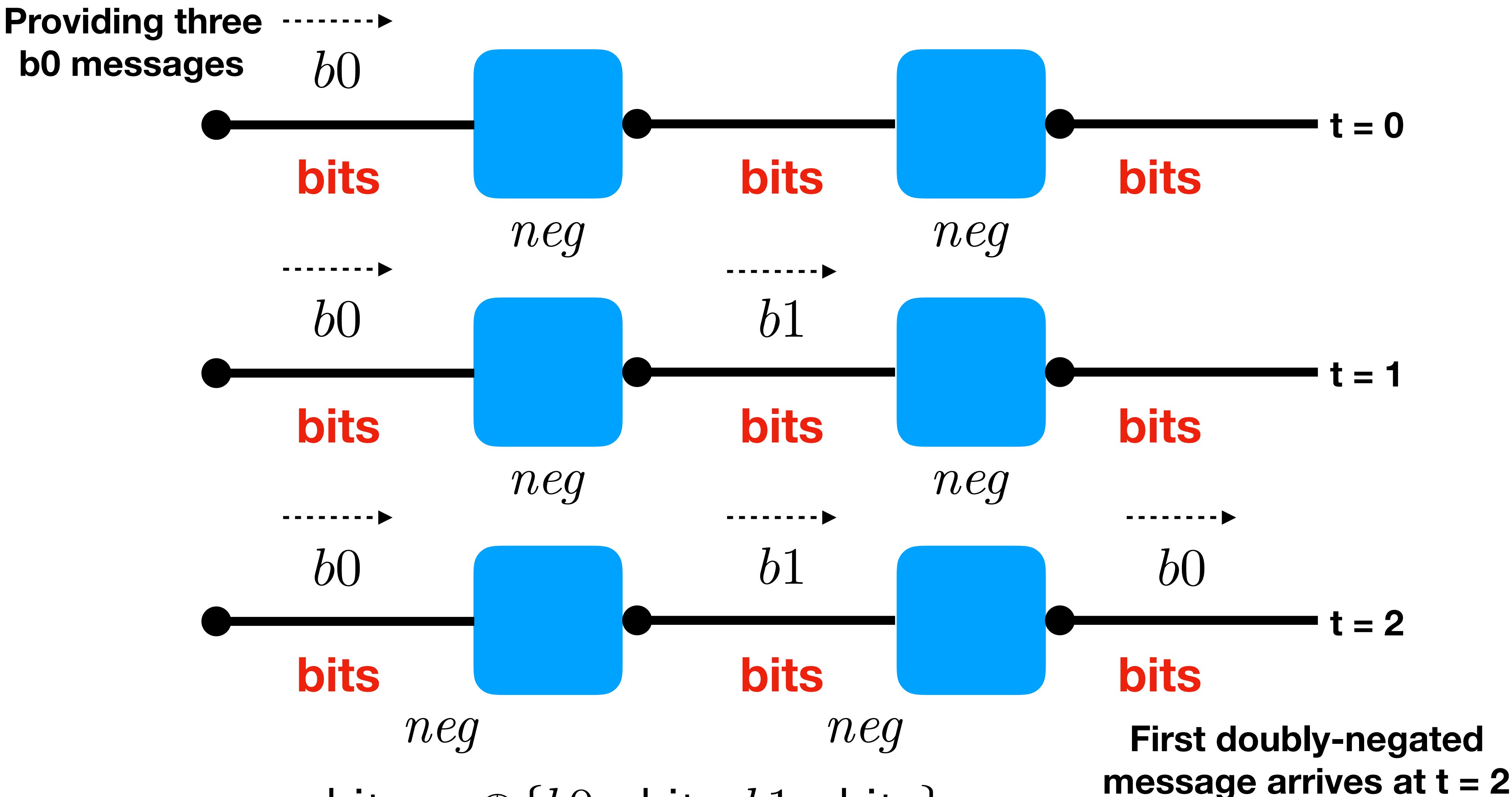
Part III
Temporal Types

Parallel Time

- Time, under the assumption of maximal parallelism
  - Every action takes place as soon as dependencies allow
- Time remains abstract, as defined by a cost model
- Examples
  - Latency in pipelines
  - Response time in interactions
  - Span in fork/join parallelism

Type System Design

- Goals same as for ergometric types!
  - Flexible: support different cost models
  - Conservative: extend rather than redefine type system or logic
  - Compositional: describe individual processes, not just whole programs
  - Precise: capture time accurately
  - General: allow many algorithms and programs to be analyzed
  - Intuitive: predictable, with good error messages
  - Automatic: infer bounds where possible



$$\mathsf{bits} = \oplus\{b0 : \mathsf{bits}, b1 : \mathsf{bits}\}$$

$$\mathsf{bits} \vdash neg : \mathsf{bits}$$

$$\begin{aligned} neg = \mathsf{caseL}\ (\, & b0 \Rightarrow \mathsf{R}.b1 \ ; \ neg \\ \mid\ & b1 \Rightarrow \mathsf{R}.b0 \ ; \ neg \,) \end{aligned}$$

Advancing Time

$$A \quad ::= \quad \ldots \mid \bigcirc A$$

$$\frac{A \vdash P : C}{\bigcirc A \vdash \textsf{tick} \; ; \; P : \bigcirc C} \; \bigcirc LR$$

All other rules remain the same! actions are cost-free, for now…

Time advances on both (all) channels simultaneously

Communication is temporally synchronized!

$$(\textsf{tick} \; ; \; P)_t \quad \longrightarrow \quad (P)_{t+1}$$

$$(\textsf{R}.k \; ; \; P)_t \mid\mid (\textsf{caseL} \; (\ell \Rightarrow Q_\ell)_{\ell \in L})_t \quad \longrightarrow \quad (P)_t \mid\mid (Q_k)_t$$

$$(\textsf{caseR} \; (\ell \Rightarrow P_\ell)_{\ell \in L})_t \mid\mid (\textsf{L}.k \; ; \; Q)_t \quad \longrightarrow \quad (P_k)_t \mid\mid (Q)_t$$

Sample Cost Model

Each receive takes 1 tick

$$
\begin{array}{lcll}
(P \mid\mid Q)^+ & = & (P)^+ \mid\mid (Q)^+ & \text{cut} \\
(\leftrightarrow)^+ & = & \leftrightarrow & \text{id} \\
(\mathsf{caseR}\,(\ell \Rightarrow P_\ell)_{\ell \in L})^+ & = & \mathsf{caseR}\,(\ell \Rightarrow \color{blue}{\mathsf{tick}}\ ;\ (P_\ell)^+)_{\ell \in L} & (\& R) \\
(\mathsf{L}.k\ ;\ Q)^+ & = & \mathsf{L}.k\ ;\ (Q)^+ & (\& L) \\
(\mathsf{R}.k\ ;\ P)^+ & = & \mathsf{R}.k\ ;\ (P)^+ & (\oplus R) \\
(\mathsf{caseL}\,(\ell \Rightarrow Q_\ell)_{\ell \in L})^+ & = & \mathsf{caseL}\,(\ell \Rightarrow \color{blue}{\mathsf{tick}}\ ;\ (Q_\ell)^+)_{\ell \in L} & (\oplus L)
\end{array}
$$

**Blue tick inserted to model cost,
not by programmer**

Timed Bit Streams

Fastest rate possible, under "receive takes 1 tick" model

$$\mathsf{bits} = \oplus\{b0 : \bigcirc\,\mathsf{bits}, b1 : \bigcirc\,\mathsf{bits}\}$$

Latency of 1 tick

$$\mathsf{bits} \vdash neg : \bigcirc\,\mathsf{bits}$$

$$neg = \mathsf{caseL}\ (\,b0 \Rightarrow \mathsf{tick}\ ;\ \mathsf{R}.b1\ ;\ neg \mid b1 \Rightarrow \mathsf{tick}\ ;\ \mathsf{R}.b0\ ;\ neg\,)$$

Latency of 2 ticks

$$\mathsf{bits} \vdash negneg : \bigcirc\,\bigcirc\,\mathsf{bits}$$

$$negneg = neg \,||\, (\mathsf{tick}\ ;\ neg)$$

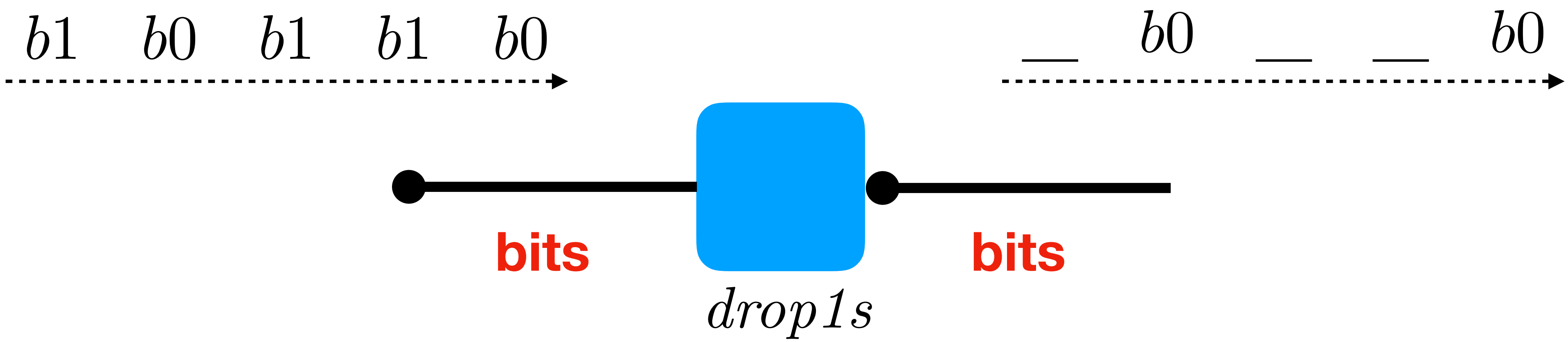
Inserted by programmer to express initial delay

No latency

$$\mathsf{bits} \vdash id : \mathsf{bits}$$

$$id = \leftrightarrow$$

Imprecision Required



$$\mathsf{bits} = \oplus\{b0 : \bigcirc\,\mathsf{bits}, b1 : \bigcirc\,\mathsf{bits}\}$$

$$\mathsf{bits} \vdash^{?} drop1s : \bigcirc\,\mathsf{bits}$$

Impossible to type in the system so far!

$$drop1s^{?} = \mathsf{caseL}\ (\,b0 \Rightarrow \mathsf{tick}\ ;\ \mathsf{R}.b0\ ;\ drop1s^{?}$$
$$\mid b1 \Rightarrow \mathsf{tick}\ ;\ drop1s^{?}\,)$$

At Some Time

Client must also be able to wait indefinitely

$$A \quad ::= \quad \ldots \mid \bigcirc A \mid \Diamond A$$

$$\frac{A \vdash P : B}{A \vdash (\textsf{now!R} \; ; \; P) : \Diamond B} \; \Diamond R \qquad \frac{B \vdash Q : C \quad (C = \bigcirc^n \Diamond C')}{\Diamond B \vdash (\textsf{when?L} \; ; \; Q) : C} \; \Diamond L$$

$$(\textsf{now!R} \; ; \; P)_t \parallel (\textsf{when?L} \; ; \; Q)_s \quad \longrightarrow \quad (P)_t \parallel (Q)_t \quad (t \geq s)$$

Updated rule(s) advancing time

$$\frac{A \vdash P : C}{\bigcirc A \vdash (\textsf{tick} \; ; \; P) : \bigcirc C} \; \bigcirc\bigcirc \qquad \frac{A \vdash P : \Diamond C}{\bigcirc A \vdash (\textsf{tick} \; ; \; P) : \Diamond C} \; \bigcirc\Diamond$$

Client continues to be ready

Irregular Rates

$$\mathsf{bits} = \oplus\{b0 : \bigcirc\,\mathsf{bits}, b1 : \bigcirc\,\mathsf{bits}\}$$

$$\mathsf{slow} = \oplus\{b0 : \bigcirc\,\Diamond\,\mathsf{slow}, b1 : \bigcirc\,\Diamond\,\mathsf{slow}\}$$

$$\mathsf{bits} \vdash \mathit{drop1s} : \bigcirc\Diamond\,\mathsf{slow}$$

$$\begin{aligned}\mathit{drop1s} = \mathsf{caseL}\ (\ & b0 \Rightarrow \mathsf{tick}\ ;\ \mathsf{now!R}\ ;\ \mathsf{R}.b0\ ;\ \mathit{drop1s} \\ \mid\ & b1 \Rightarrow \mathsf{tick}\ ;\ \mathit{drop1s} \,\|\, \mathit{idle}\ )\end{aligned}$$

$\mathsf{bits} \vdash \Diamond\,\mathsf{slow}$

$\bigcirc\,\Diamond\,\mathsf{slow}$

$$\bigcirc\,\Diamond\,\mathsf{slow} \vdash \mathit{idle} : \Diamond\,\mathsf{slow}$$

$$\mathit{idle} = \mathsf{tick}\ ;\ \leftrightarrow$$

Time reconstruction can infer the idle process by subtyping

At All Times

$$A \quad ::= \quad \ldots \mid \bigcirc A \mid \Diamond A \mid \Box A$$

"Always A" or "Box A"
Rules are symmetric to $\Diamond$A

Response Times

$$A \quad ::= \quad \ldots \mid \bigcirc A \mid \Diamond A \mid \Box A$$

$$\mathsf{bits} = \oplus\{b0 : \bigcirc\, \mathsf{bits}, b1 : \bigcirc\, \mathsf{bits}, \$: \bigcirc\, \mathbf{1}\}$$

$$\mathsf{ctr} = \&\{inc : \bigcirc\, \Box\, \mathsf{ctr}, val : \bigcirc\, \mathsf{bits}\}$$

Response time of 1 tick in each case

$$\begin{array}{rcccl} & \vdash & empty & : & \Box\, \mathsf{ctr} \\ \bigcirc\, \Box\, \mathsf{ctr} & \vdash & bit0 & : & \Box\, \mathsf{ctr} \\ \Box\, \mathsf{ctr} & \vdash & bit1 & : & \Box\, \mathsf{ctr} \end{array}$$

Increment or read out value at any time (client's choice)

Temporal Types

- ☑ Flexible: support different cost models
- ☑ Conservative: extend rather than redefine type system or logic
- ☑ Compositional: describe individual processes, not just whole programs
- ☑ Precise: capture time accurately
- ■ General: allow many algorithms and programs to be analyzed
- ■ Intuitive: predictable, with good error messages
- ■ Automatic: infer bounds where possible

More Examples

In a cost model where both send and receive take 1 tick

Queue as bucket brigade

$$\mathsf{queue}_A = \&\{\ ins : \bigcirc (\Box A \multimap \bigcirc^3 \Box\, \mathsf{queue}_A),\ del : \bigcirc \oplus \{\ none : \bigcirc \mathbf{1},\ some : \bigcirc (\Box A \otimes \bigcirc \Box\, \mathsf{queue}_A)\ \}\ \}$$

$$\mathsf{list}^r_A[n] = \oplus\{\ nil : \exists\{n = 0\} \bigcirc \mathbf{1},\ cons : \exists\{n > 0\} \bigcirc (\Box A \otimes \bigcirc \bigcirc^{r+2} \mathsf{list}_A[n-1])\ \}$$

$$(l_1 : \mathsf{list}^r_A[n])\ (l_2 : \bigcirc^{(r+4)n+2} \mathsf{list}^r_A[k]) \vdash append :: (l : \bigcirc \bigcirc \mathsf{list}^r_A[n+k])$$

$$\mathsf{stream}^r_A = \Box A \otimes \bigcirc \bigcirc^r \mathsf{stream}^r_A$$

$$(l_1 : \mathsf{stream}^3_A)\ (l_2 : \bigcirc^2 \mathsf{stream}^3_A) \vdash alternate :: (l : \bigcirc \mathsf{stream}^1_A)$$

$$(l_1 : \mathsf{stream}^{2r+3}_A)\ (l_2 : \bigcirc^{r+2} \mathsf{stream}^{2r+3}_A) \vdash alternate :: (l : \bigcirc \mathsf{stream}^{r+1}_A)$$

Theory

- Logically, a form of “linear-time linear logic”
- Preservation and progress follow (some complexities)
- Communication can be temporally synchronized (depending on the cost model and implementation)
- Compatible with ergometric types
- Subtyping and time reconstruction

Summary

$\oplus\{\ell : A_\ell\}_{\ell \in L}$	send $k \in L$	cont as A_k
$\&\{\ell : A_\ell\}_{\ell \in L}$	recv $k \in L$	cont as A_k
$\mathbf{1}$	send end	(terminate)
$\triangleright^r A$	send potential r	cont as A
$\triangleleft^r A$	recv potential r	cont as A
$\bigcirc A$	delay for 1 tick	cont as A
$\Diamond A$	send now at some time	cont as A
$\Box A$	recv now at some time	cont as A
$A \multimap B$	recv channel $c : A$	cont as B
$A \otimes B$	send channel $c : A$	cont as B

Take-Aways

- Session types prescribe bidirectional communication protocols along channels with two endpoints (provider and client) ↔ (intuitionistic) linear logic

- There are elegant and expressive conservative extensions to capture work (e.g., total messages sent) and parallel time ↔ temporal linear logic

- Work and time reconstruction for modularity and brevity

- Ongoing: implementation, parametric resource inference

Some Acknowledgments

- *Logical foundations of session types*, several papers with multiple collaborators (Luís Caires, Bernardo Toninho, Jorge A. Pérez, Dennis Griffith, Klaas Pruiksma)
- *Work Analysis with Resource-Aware Session Types* (with Ankush Das & Jan Hoffmann, LICS 2018)
- *Parallel Complexity with Temporal Session Types* (with Ankush Das & Jan Hoffmann, ICFP 2018)*