# Verificación de Programas Concurrentes Usando Lógica
# *Logical Foundations of Concurrent Computation*

Jorge A. Pérez
University of Groningen, The Netherlands
j.a.perez [[at]] rug.nl

ECI 2026
(Version of July 27, 2026)

# Preambulo

# Outline

# Verificación de Programas Concurrentes

- Muchos sistemas informáticos son **concurrentes**, y están compuestos de programas que se ejecutan simultáneamente.

# Verificación de Programas Concurrentes

- Muchos sistemas informáticos son **concurrentes**, y están compuestos de programas que se ejecutan simultáneamente.
- **Ejemplos:**
  - Sistemas distribuidos y paralelos
  - Arquitecturas basadas en microservicios
  - Go / Erlang / Rust / Akka / Elixir

# Verificación de Programas Concurrentes

- Muchos sistemas informáticos son **concurrentes**, y están compuestos de programas que se ejecutan simultáneamente.
- **Ejemplos:**
  - Sistemas distribuidos y paralelos
  - Arquitecturas basadas en microservicios
  - Go / Erlang / Rust / Akka / Elixir
- Mecanismo de coordinación entre programas: **paso de mensajes**.

# Verificación de Programas Concurrentes

- Muchos sistemas informáticos son **concurrentes**, y están compuestos de programas que se ejecutan simultáneamente.
- **Ejemplos:**
  - Sistemas distribuidos y paralelos
  - Arquitecturas basadas en microservicios
  - Go / Erlang / Rust / Akka / Elixir
- Mecanismo de coordinación entre programas: **paso de mensajes**.
- Establecer la correctitud de estos programas implica garantizar que las interacciones entre ellos respetan los **protocolos establecidos**.

# Verificación de Programas Concurrentes

- Muchos sistemas informáticos son **concurrentes**, y están compuestos de programas que se ejecutan simultáneamente.
- **Ejemplos:**
  - Sistemas distribuidos y paralelos
  - Arquitecturas basadas en microservicios
  - Go / Erlang / Rust / Akka / Elixir
- Mecanismo de coordinación entre programas: **paso de mensajes**.
- Establecer la correctitud de estos programas implica garantizar que las interacciones entre ellos respetan los **protocolos establecidos**.
- Una tarea crucial, pero también compleja: necesitamos técnicas de verificación capaces de analizar **estructuras de comunicación** sofisticadas.

# Verificación de Programas Concurrentes **Usando Lógica**

**Este Curso**

- Técnicas fundamentales para la verificación para programas concurrentes.
- **Enfasis:** Conceptos de la **lógica** para el análisis y verificación de programas.

# Verificación de Programas Concurrentes **Usando Lógica**

## Este Curso

- Técnicas fundamentales para la verificación para programas concurrentes.
- **Enfasis:** Conceptos de la **lógica** para el análisis y verificación de programas.

## Ideas Principales

- Describir los protocolos de comunicación usando **sistemas de tipos**.
- Especificar los programas concurrentes con un **lenguaje formal de procesos**.
- Los tipos aseguran que los procesos **usan sus recursos correctamente**.

$\rightarrow$ Uso adecuado de recursos = ejecución correcta de protocolos.

# Verificación de Programas Concurrentes **Usando Lógica**

## Este Curso

- Técnicas fundamentales para la verificación para programas concurrentes.
- **Enfasis:** Conceptos de la **lógica** para el análisis y verificación de programas.

## Ideas Principales

- Describir los protocolos de comunicación usando **sistemas de tipos**.
- Especificar los programas concurrentes con un **lenguaje formal de procesos**.
- Los tipos aseguran que los procesos **usan sus recursos correctamente**.

$\rightarrow$ Uso adecuado de recursos = ejecución correcta de protocolos.

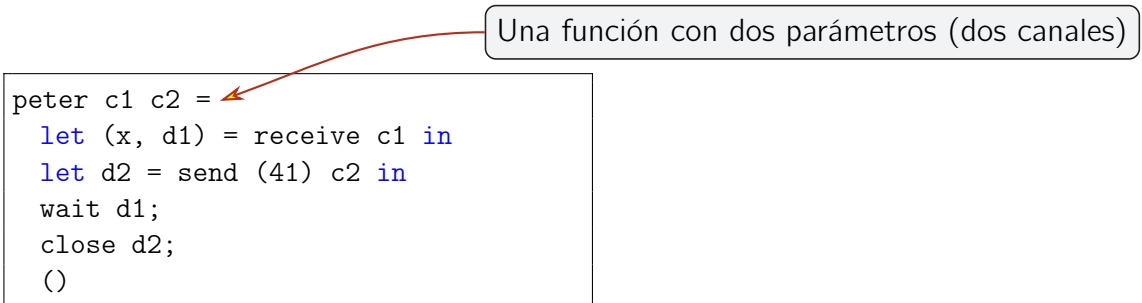
## Correspondencia de Curry-Howard

Demostrar un teorema y escribir un programa correcto son la **misma actividad** desde perspectivas distintas.

# Ejemplo: Un Programa con Paso de Mensajes

```
peter c1 c2 =
  let (x, d1) = receive c1 in
  let d2 = send (41) c2 in
  wait d1;
  close d2;
  ()
```

# Ejemplo: Un Programa con Paso de Mensajes



# Ejemplo: Un Programa con Paso de Mensajes

```
peter c1 c2 =
  let (x, d1) = receive c1 in
  let d2 = send (41) c2 in
  wait d1;
  close d2;
  ()
```

Recibe un mensaje en c1, continúa en d1

# Ejemplo: Un Programa con Paso de Mensajes

```
peter c1 c2 =
  let (x, d1) = receive c1 in
  let d2 = send (41) c2 in
  wait d1;
  close d2;
  ()
```

Envía ‘41’ en c2, continúa en d2

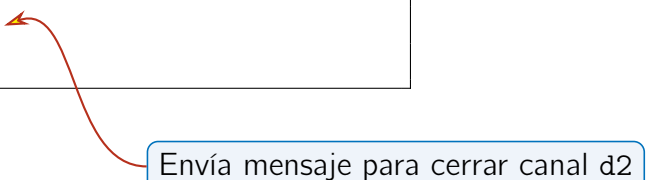
# Ejemplo: Un Programa con Paso de Mensajes

```
peter c1 c2 =
  let (x, d1) = receive c1 in
  let d2 = send (41) c2 in
  wait d1;
  close d2;
  ()
```

Espera confirmación para cerrar canal d1

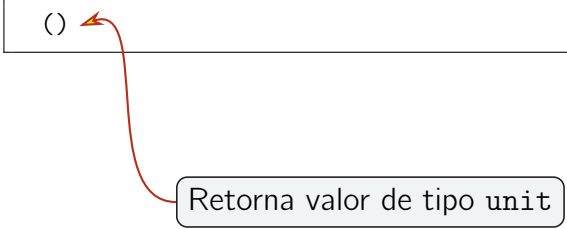
# Ejemplo: Un Programa con Paso de Mensajes

```
peter c1 c2 =
  let (x, d1) = receive c1 in
  let d2 = send (41) c2 in
  wait d1;
  close d2;
  ()
```



# Ejemplo: Un Programa con Paso de Mensajes

```
peter c1 c2 =
  let (x, d1) = receive c1 in
  let d2 = send (41) c2 in
  wait d1;
  close d2;
  ()
```



# Ejemplo: Dos Programas con Paso de Mensajes

```
peter c1 c2 =
  let (x, c1) = receive c1 in
  let c2 = send (41) c2 in
  wait c1;
  close c2;
  ()
```

```
miles c1 c2 =
  let (x, c2) = receive c2 in
  let c1 = send (42) c1 in
  wait c2;
  close c1;
  ()
```

# Ejemplo: Dos Programas con Paso de Mensajes

```
peter c1 c2 =
  let (x, c1) = receive c1 in
  let c2 = send (41) c2 in
  wait c1;
  close c2;
  ()
```

El programa respeta su protocolo ✔

```
miles c1 c2 =
  let (x, c2) = receive c2 in
  let c1 = send (42) c1 in
  wait c2;
  close c1;
  ()
```

# Ejemplo: Dos Programas con Paso de Mensajes

```
peter c1 c2 =
  let (x, c1) = receive c1 in
  let c2 = send (41) c2 in
  wait c1;
  close c2;
  ()
```

El programa respeta su protocolo ✔

```
miles c1 c2 =
  let (x, c2) = receive c2 in
  let c1 = send (42) c1 in
  wait c2;
  close c1;
  ()
```

El programa también respeta su protocolo ✔

# Ejemplo: Interacción de Programas

```
peter c1 c2 =
  let (x, c1) = receive c1 in
  let c2 = send (41) c2 in
  wait c1;
  close c2;
  ()
```

```
miles c1 c2 =
  let (x, c2) = receive c2 in
  let c1 = send (42) c1 in
  wait c2;
  close c1;
  ()
```

```
main =
  let (c1, c1dual) = new @T () in
  let (c2, c2dual) = new @T () in
  fork (miles c1dual c2);
       peter c1 c2dual
```

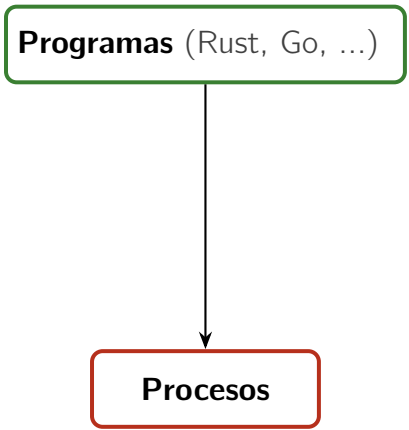
Es 'main' correcto?

# Verificación de Programas usando Sistemas de Tipos

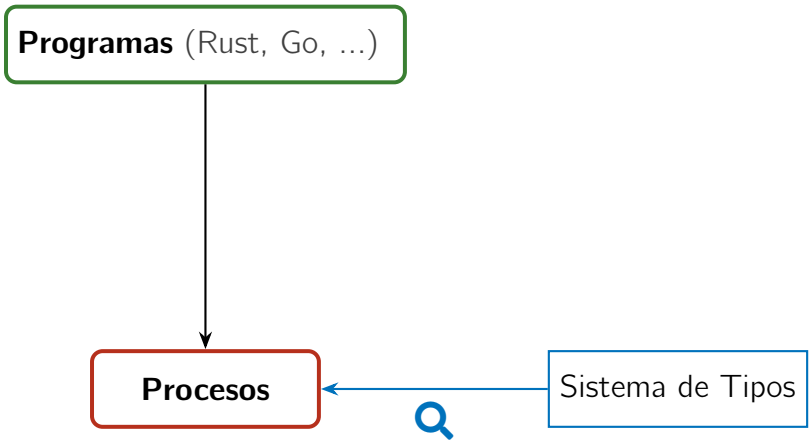
**Programas** (Rust, Go, ...)

**Procesos**

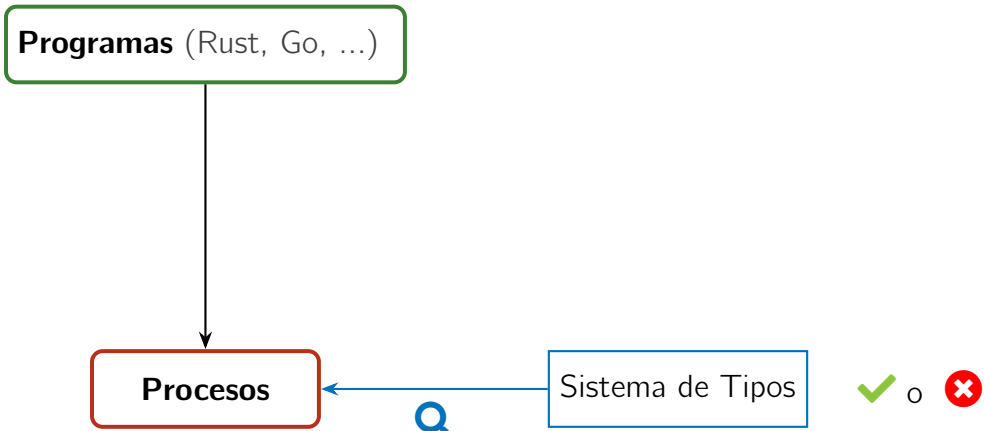
# Verificación de Programas usando Sistemas de Tipos



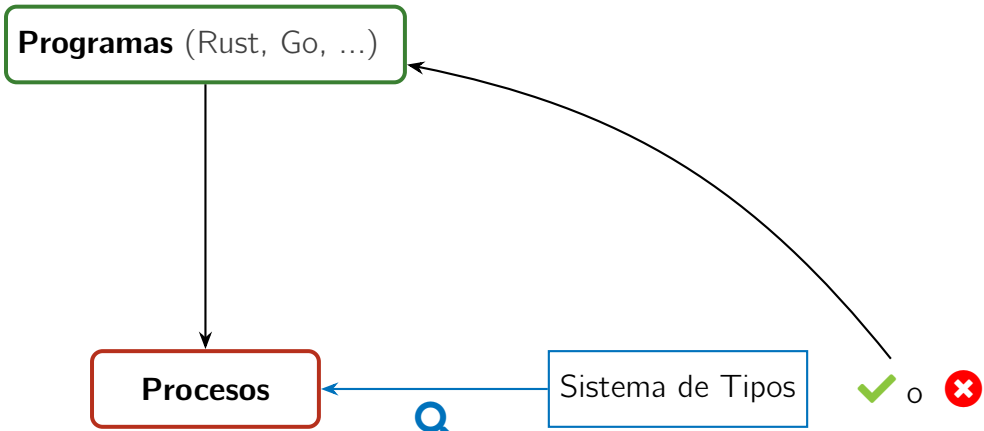
# Verificación de Programas usando Sistemas de Tipos



# Verificación de Programas usando Sistemas de Tipos



# Verificación de Programas usando Sistemas de Tipos



# Outline

# Un Lenguaje Formal para Concurrencia: Cálculos de Procesos

Asumiendo un conjunto de **nombres** $x, y, z, \ldots$ (usados para representar **canales**), definimos un lenguaje formal de **procesos** (denotados $P, Q, \ldots$):

# Un Lenguaje Formal para Concurrencia: Cálculos de Procesos

Asumiendo un conjunto de **nombres** $x, y, z, \ldots$ (usados para representar **canales**), definimos un lenguaje formal de **procesos** (denotados $P, Q, \ldots$):

$$
\begin{array}{rcll}
P, Q ::= & \overline{x}\langle y \rangle.P & \textbf{envia} \text{ mensaje } y \text{ en el canal } x, \text{ continúa con } P \\
\mid & x(y).P & \textbf{recibe} \text{ un mensaje } z \text{ en el canal } x, \text{ continúa con } P\{^z/_y\}
\end{array}
$$

# Un Lenguaje Formal para Concurrencia: Cálculos de Procesos

Asumiendo un conjunto de **nombres** $x, y, z, \ldots$ (usados para representar **canales**), definimos un lenguaje formal de **procesos** (denotados $P, Q, \ldots$):

$$
\begin{array}{rll}
P, Q ::= & \overline{x}\langle y \rangle.P & \textbf{envia} \text{ mensaje } y \text{ en el canal } x, \text{ continúa con } P \\
\mid & x(y).P & \textbf{recibe} \text{ un mensaje } z \text{ en el canal } x, \text{ continúa con } P\{^z/_y\} \\
\mid & \overline{x}\langle\rangle.P & \textbf{envía} \text{ una señal para cerrar el canal } x, \text{ continúa como } P \\
\mid & x().P & \textbf{recibe} \text{ señal para cerrar el canal } x, \text{ continúa como } P
\end{array}
$$

# Un Lenguaje Formal para Concurrencia: Cálculos de Procesos

Asumiendo un conjunto de **nombres** $x, y, z, \ldots$ (usados para representar **canales**), definimos un lenguaje formal de **procesos** (denotados $P, Q, \ldots$):

| | | |
|---|---|---|
| $P, Q ::=$ | $\overline{x}\langle y\rangle.P$ | **envia** mensaje $y$ en el canal $x$, continúa con $P$ |
| $\mid$ | $x(y).P$ | **recibe** un mensaje $z$ en el canal $x$, continúa con $P\{^z/_y\}$ |
| $\mid$ | $\overline{x}\langle\rangle.P$ | **envía** una señal para cerrar el canal $x$, continúa como $P$ |
| $\mid$ | $x().P$ | **recibe** señal para cerrar el canal $x$, continúa como $P$ |
| $\mid$ | $P \mid Q$ | **concurrencia**: $P$ y $Q$ se ejecutan en paralelo |
| $\mid$ | $(\boldsymbol{\nu} x)P$ | **restricción**: el canal $x$ es local (privado) en $P$ |
| $\mid$ | $0$ | **cero**: el proceso que no hace nada |

# Un Lenguaje Formal: Cálculos de Procesos

Sintaxis de procesos:

$$P, Q ::= \overline{x}\langle y\rangle.P \mid x(y).P \mid \overline{x}\langle\rangle.P \mid x().P \mid P \mid Q \mid (\nu x)P \mid 0$$

Cómo expresar interacción?

# Un Lenguaje Formal: Cálculos de Procesos

Sintaxis de procesos:

$$P, Q ::= \overline{x}\langle y\rangle.P \mid x(y).P \mid \overline{x}\langle\rangle.P \mid x().P \mid P \mid Q \mid (\nu x)P \mid 0$$

Cómo expresar interacción?
La **relación de reducción** formaliza la comunicación entre procesos concurrentes.
Dos reglas de la definición:

$$\overline{x}\langle z\rangle.P \mid x(y).Q \longrightarrow P \mid Q\{^z/_y\}$$

$$\overline{x}\langle\rangle.P \mid x().Q \longrightarrow P \mid Q$$

# Un Lenguaje Formal: Cálculos de Procesos

Sintaxis de procesos:

$$P, Q ::= \overline{x}\langle y\rangle.P \;\Big|\; x(y).P \;\Big|\; \overline{x}\langle\rangle.P \;\Big|\; x().P \;\Big|\; P \mid Q \;\Big|\; (\nu x)P \;\Big|\; 0$$

Cómo expresar interacción?
La **relación de reducción** formaliza la comunicación entre procesos concurrentes.
Dos reglas de la definición:

$$\overline{x}\langle z\rangle.P \mid x(y).Q \longrightarrow P \mid Q\{^z/_y\}$$

$$\overline{x}\langle\rangle.P \mid x().Q \longrightarrow P \mid Q$$

**Intuición:**
Acciones que ocurren en paralelo (y en el mismo canal) pueden dar lugar a una sincronización. Las acciones deben ser complementarias (input / output).

# Ejercicio: Cuáles procesos pueden reducir (interactuar)?

1) $x(z).P \mid x(u).Q$

2) $\overline{x}\langle z\rangle.P \mid z(u).Q$

3) $\overline{x}\langle\rangle.x().P$

4) $\overline{x}\langle v_1\rangle.x(z).P \mid x(y).\overline{x}\langle v_2\rangle.Q$

5) $\overline{x}\langle z\rangle.0 \mid x(y).y(u).Q_1 \mid x(y).y().0$

# Ejercicio: Cuáles procesos pueden reducir (interactuar)?

1) $x(z).P \mid x(u).Q$

no hay reducción:

las acciones no son complementarias

2) $\overline{x}\langle z\rangle.P \mid z(u).Q$

3) $\overline{x}\langle\rangle.x().P$

4) $\overline{x}\langle v_1\rangle.x(z).P \mid x(y).\overline{x}\langle v_2\rangle.Q$

5) $\overline{x}\langle z\rangle.0 \mid x(y).y(u).Q_1 \mid x(y).y().0$

# Ejercicio: Cuáles procesos pueden reducir (interactuar)?

1) $x(z).P \mid x(u).Q$

no hay reducción:
las acciones no son complementarias

2) $\overline{x}\langle z\rangle.P \mid z(u).Q$

no hay reducción:
los canales son diferentes

3) $\overline{x}\langle\rangle.x().P$

4) $\overline{x}\langle v_1\rangle.x(z).P \mid x(y).\overline{x}\langle v_2\rangle.Q$

5) $\overline{x}\langle z\rangle.0 \mid x(y).y(u).Q_1 \mid x(y).y().0$

# Ejercicio: Cuáles procesos pueden reducir (interactuar)?

1) $x(z).P \mid x(u).Q$

no hay reducción:
las acciones no son complementarias

2) $\overline{x}\langle z\rangle.P \mid z(u).Q$

no hay reducción:
los canales son diferentes

3) $\overline{x}\langle\rangle.x().P$

no hay reducción:
las acciones son secuenciales

4) $\overline{x}\langle v_1\rangle.x(z).P \mid x(y).\overline{x}\langle v_2\rangle.Q$

5) $\overline{x}\langle z\rangle.0 \mid x(y).y(u).Q_1 \mid x(y).y().0$

# Ejercicio: Cuáles procesos pueden reducir (interactuar)?

1) $x(z).P \mid x(u).Q$ — no hay reducción: las acciones no son complementarias

2) $\overline{x}\langle z\rangle.P \mid z(u).Q$ — no hay reducción: los canales son diferentes

3) $\overline{x}\langle\rangle.x().P$ — no hay reducción: las acciones son secuenciales

4) $\overline{x}\langle v_1\rangle.x(z).P \mid x(y).\overline{x}\langle v_2\rangle.Q$ — hay dos reducciones

5) $\overline{x}\langle z\rangle.0 \mid x(y).y(u).Q_1 \mid x(y).y().0$

# Ejercicio: Cuáles procesos pueden reducir (interactuar)?

| | |
|---|---|
| 1) $x(z).P \mid x(u).Q$ | no hay reducción:<br>las acciones no son complementarias |
| 2) $\overline{x}\langle z\rangle.P \mid z(u).Q$ | no hay reducción:<br>los canales son diferentes |
| 3) $\overline{x}\langle\rangle.x().P$ | no hay reducción:<br>las acciones son secuenciales |
| 4) $\overline{x}\langle v_1\rangle.x(z).P \mid x(y).\overline{x}\langle v_2\rangle.Q$ | hay dos reducciones |
| 5) $\overline{x}\langle z\rangle.0 \mid x(y).y(u).Q_1 \mid x(y).y().0$ | hay una reducción:<br>el canal $z$ es un mensaje! |

# Ejercicio: Cuáles procesos pueden reducir (interactuar)?

Veamos (4) en detalle:

$$\begin{aligned} \overline{x}\langle v_1\rangle.x(z).P \mid x(y).\overline{x}\langle v_2\rangle.Q &\longrightarrow x(z).P \mid \overline{x}\langle v_2\rangle.Q\{^{v_1}/_{y}\} \\ &\longrightarrow P\{^{v_2}/_{z}\} \mid Q\{^{v_1}/_{y}\} \end{aligned}$$

# Ejercicio: Cuáles procesos pueden reducir (interactuar)?

Veamos (4) en detalle:

$$\begin{aligned} \overline{x}\langle v_1\rangle.x(z).P \mid x(y).\overline{x}\langle v_2\rangle.Q &\longrightarrow x(z).P \mid \overline{x}\langle v_2\rangle.Q\{^{v_1}/_{y}\} \\ &\longrightarrow P\{^{v_2}/_{z}\} \mid Q\{^{v_1}/_{y}\} \end{aligned}$$

(5) es interesante: hay dos posibles reducciones, pero sólo una puede ocurrir. Por un lado tenemos:

$$\overline{x}\langle z\rangle.0 \mid x(y).y(u).Q_1 \mid x(y).y().0 \longrightarrow 0 \mid z(u).Q_1\{^{z}/_{y}\} \mid x(y).y().0$$

# Ejercicio: Cuáles procesos pueden reducir (interactuar)?

Veamos (4) en detalle:

$$\begin{aligned}\overline{x}\langle v_1\rangle.x(z).P \mid x(y).\overline{x}\langle v_2\rangle.Q &\longrightarrow x(z).P \mid \overline{x}\langle v_2\rangle.Q\{^{v_1}/_{y}\} \\ &\longrightarrow P\{^{v_2}/_{z}\} \mid Q\{^{v_1}/_{y}\}\end{aligned}$$

(5) es interesante: hay dos posibles reducciones, pero sólo una puede ocurrir. Por un lado tenemos:

$$\overline{x}\langle z\rangle.0 \mid x(y).y(u).Q_1 \mid x(y).y().0 \longrightarrow 0 \mid z(u).Q_1\{^{z}/_{y}\} \mid x(y).y().0$$

pero también:

$$\overline{x}\langle z\rangle.0 \mid x(y).y(u).Q_1 \mid x(y).y().0 \longrightarrow 0 \mid x(y).y(u).Q_1 \mid z().0$$

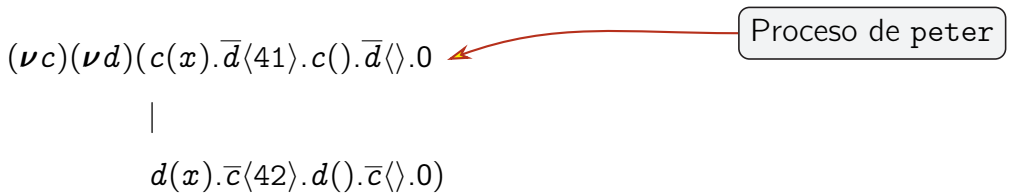
# Un Lenguaje Formal: Cálculos de Procesos

Regresando al programa `main`, el protocolo de comunicación correspondiente se puede expresar de forma formal y compacta usando procesos:

$$(\boldsymbol{\nu} c)(\boldsymbol{\nu} d)(c(x).\overline{d}\langle 41\rangle.c().\overline{d}\langle\rangle.0$$
$$\mid$$
$$d(x).\overline{c}\langle 42\rangle.d().\overline{c}\langle\rangle.0)$$

# Un Lenguaje Formal: Cálculos de Procesos

Regresando al programa `main`, el protocolo de comunicación correspondiente se puede expresar de forma formal y compacta usando procesos:



# Un Lenguaje Formal: Cálculos de Procesos

Regresando al programa `main`, el protocolo de comunicación correspondiente se puede expresar de forma formal y compacta usando procesos:

$$(\boldsymbol{\nu} c)(\boldsymbol{\nu} d)(c(x).\overline{d}\langle 41\rangle.c().\overline{d}\langle\rangle.0$$

$$\mid$$

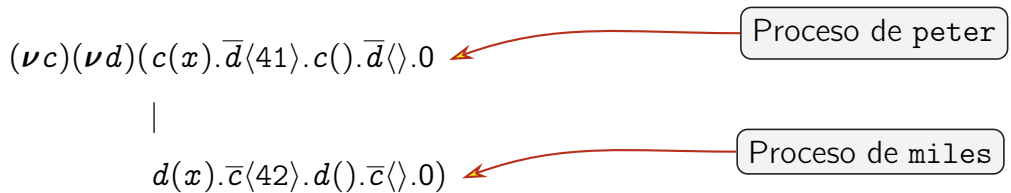
$$d(x).\overline{c}\langle 42\rangle.d().\overline{c}\langle\rangle.0)$$

Proceso de `peter`

Proceso de `miles`

# Un Lenguaje Formal: Cálculos de Procesos

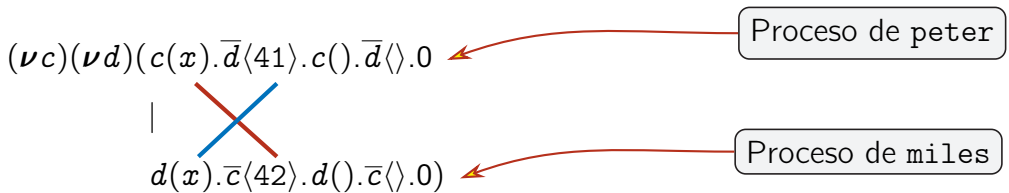
Regresando al programa `main`, el protocolo de comunicación correspondiente se puede expresar de forma formal y compacta usando procesos:




El análisis del sistema completo nos permite detectar una dependencia circular entre los dos sub-procesos: el programa no puede reducir y llega a un **deadlock**.

# Un Lenguaje Formal: Cálculos de Procesos

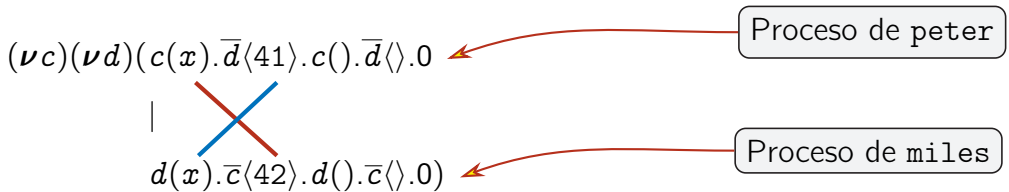
Regresando al programa `main`, el protocolo de comunicación correspondiente se puede expresar de forma formal y compacta usando procesos:




El análisis del sistema completo nos permite detectar una dependencia circular entre los dos sub-procesos: el programa no puede reducir y llega a un **deadlock**.

# Un Lenguaje Formal: Cálculos de Procesos

Regresando al programa `main`, el protocolo de comunicación correspondiente se puede expresar de forma formal y compacta usando procesos:




El análisis del sistema completo nos permite detectar una dependencia circular entre los dos sub-procesos: el programa no puede reducir y llega a un **deadlock**.

**Este curso**

Presentamos el uso de lógica para la verificación estática de procesos concurrentes, asegurando que los protocolos se ejecutan correctamente, sin llegar a *deadlocks*.

# Outline

# Plan del Curso

1. **Session types**: Especificaciones precisas de protocolos para canales.

# Plan del Curso

1. **Session types**: Especificaciones precisas de protocolos para canales.
2. **Linear Logic (LL)**: Una lógica para el uso correcto de recursos computacionales.

# Plan del Curso

1. **Session types**: Especificaciones precisas de protocolos para canales.
2. **Linear Logic (LL)**: Una lógica para el uso correcto de recursos computacionales.
3. La **correspondencia de Curry-Howard** para procesos concurrentes:

$$\begin{aligned} \text{proposiciones en LL} &\iff \text{session types} \\ \text{pruebas en LL} &\iff \text{procesos concurrentes} \\ \text{simplificación de pruebas} &\iff \text{sincronización entre procesos} \end{aligned}$$

# Plan del Curso

1. **Session types**: Especificaciones precisas de protocolos para canales.
2. **Linear Logic (LL)**: Una lógica para el uso correcto de recursos computacionales.
3. La **correspondencia de Curry-Howard** para procesos concurrentes:

$$\begin{aligned} \text{proposiciones en LL} &\iff \text{session types} \\ \text{pruebas en LL} &\iff \text{procesos concurrentes} \\ \text{simplificación de pruebas} &\iff \text{sincronización entre procesos} \end{aligned}$$

4. La correspondencia usando LL en su **versión clásica**.

# Plan del Curso

1. **Session types**: Especificaciones precisas de protocolos para canales.
2. **Linear Logic (LL)**: Una lógica para el uso correcto de recursos computacionales.
3. La **correspondencia de Curry-Howard** para procesos concurrentes:

$$\begin{aligned} \text{proposiciones en LL} &\iff \text{session types} \\ \text{pruebas en LL} &\iff \text{procesos concurrentes} \\ \text{simplificación de pruebas} &\iff \text{sincronización entre procesos} \end{aligned}$$

4. La correspondencia usando LL en su **versión clásica**.
5. Comunicación asíncrona y deadlock freedom.

---

# Plan del Curso

1. **Session types**: Especificaciones precisas de protocolos para canales.
2. **Linear Logic (LL)**: Una lógica para el uso correcto de recursos computacionales.
3. La **correspondencia de Curry-Howard** para procesos concurrentes:

$$\begin{aligned} \text{proposiciones en LL} &\iff \text{session types} \\ \text{pruebas en LL} &\iff \text{procesos concurrentes} \\ \text{simplificación de pruebas} &\iff \text{sincronización entre procesos} \end{aligned}$$

4. La correspondencia usando LL en su **versión clásica**.
5. Comunicación asíncrona y deadlock freedom.

---

6. Extensión de la correspondencia con recursos ilimitados: **clientes y servidores**.
7. La lógica de **implicaciones agrupadas** (*bunched implications*).

# Aspectos Prácticos

Cada clase estará dividida en tres partes:

- 1:30 - 2:15: Parte 1
- 2:15 - 2:20: Pausa
- 2:20 - 3:00: Parte 2
- 3:00 - 3:20: Coffee break
- 3:20 - 4:30: Parte 3

Material asociado al curso, actualizado después de cada clase:

`https://jperez.nl/teaching/eci26`

Preguntas, sugerencias, consultas:

`j.a.perez@rug.nl` (subject: [eci26])

# Session Types

# This Course: Session Types / Propositions as Sessions

An introduction to **session types** for concurrency, and to the logical foundations of message-passing computation.

# This Course: Session Types / Propositions as Sessions

An introduction to **session types** for concurrency, and to the logical foundations of message-passing computation.

- In a nutshell, a **session** provides a structure for a series of related interactions between communicating programs.
  We often talk about **session protocols**.

# This Course: Session Types / Propositions as Sessions

An introduction to **session types** for concurrency, and to the logical foundations of message-passing computation.

- In a nutshell, a **session** provides a structure for a series of related interactions between communicating programs.
  We often talk about **session protocols**.
- A disciplined usage of sessions is key to ensure program correctness.
  Not only: sessions have elegant **logical foundations**!

# This Course: Session Types / Propositions as Sessions

An introduction to **session types** for concurrency, and to the logical foundations of message-passing computation.

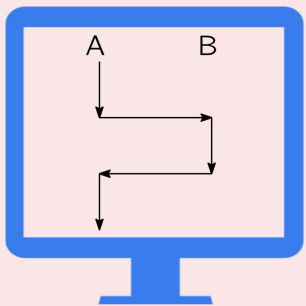
- In a nutshell, a **session** provides a structure for a series of related interactions between communicating programs.
  We often talk about **session protocols**.
- A disciplined usage of sessions is key to ensure program correctness.
  Not only: sessions have elegant **logical foundations**!
- **Linear Logic (LL)**: A resource-oriented view on propositions / protocols.
  Linear resources can be used exactly once.

In the rest of the lecture: the language of session types, and basics of linear logic.

# Outline

# When is a Program Correct?

## Sequential Programs




"Programs produce outputs that are consistent with their input"

# Concurrent Programs?

# Message-Passing Concurrent Programs?

- Software components (**services**) distributed across networks

# Message-Passing Concurrent Programs

- Software components (**services**) distributed across networks
- Coordination takes place by sending and receiving messages

# Message-Passing Concurrent Programs

- Software components (**services**) distributed across networks
- Coordination takes place by sending and receiving messages
- Compatible message exchanges are crucial for system correctness

# Message-Passing Concurrent Programs

- Software components (**services**) distributed across networks
- Coordination takes place by sending and receiving messages
- Compatible message exchanges are crucial for system correctness
- Even a single faulty exchange can cause system-wide bugs

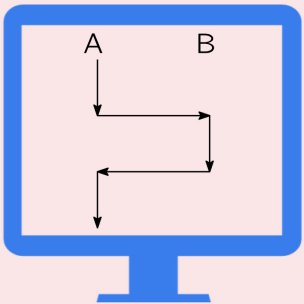
# Message-Passing Concurrent Programs

- Software components (**services**) distributed across networks
- Coordination takes place by sending and receiving messages
- Compatible message exchanges are crucial for system correctness
- Even a single faulty exchange can cause system-wide bugs

**An (imperfect) analogy**:

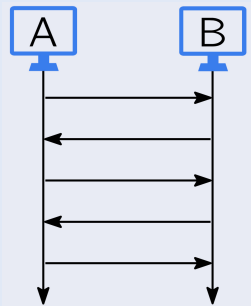
# When is a Program Correct?

## Sequential Programs



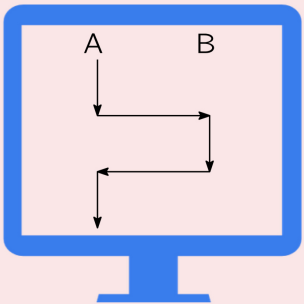

"Programs produce outputs that are consistent with their input"

## Concurrent Programs



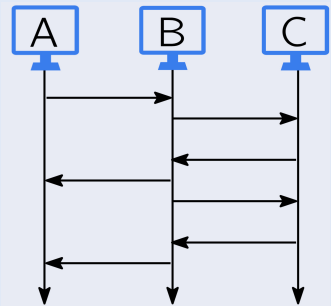
# When is a Program Correct?

## Sequential Programs



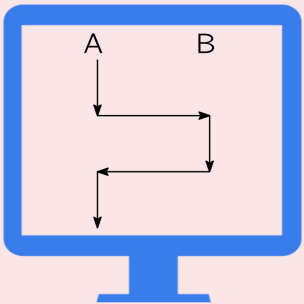

"Programs produce outputs that are consistent with their input"

## Concurrent Programs



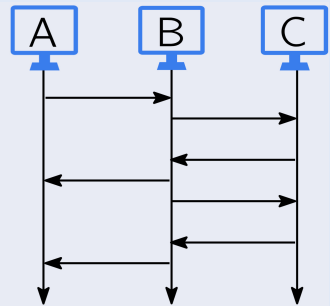
# When is a Program Correct?

## Sequential Programs




"Programs produce outputs that are consistent with their input"

## Concurrent Programs

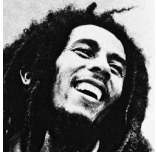



"Programs always respect their intended **protocols**"

# Example: A Two-Buyer Protocol


**Alice** and **Bob** cooperate in buying a book from **Seller**

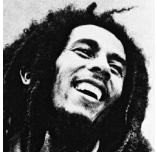
# Example: A Two-Buyer Protocol




**Alice** and **Bob** cooperate in buying a book from **Seller** :

1. Alice sends a book title to Seller, who sends a quote back.

# Example: A Two-Buyer Protocol



**Alice** and **Bob** cooperate in buying a book from **Seller** :

1. Alice sends a book title to Seller, who sends a quote back.
2. Alice checks whether Bob can contribute in buying the book.

# Example: A Two-Buyer Protocol

**Alice** and **Bob** cooperate in buying a book from **Seller** :

1. Alice sends a book title to Seller, who sends a quote back.
2. Alice checks whether Bob can contribute in buying the book.
3. Alice uses the answer from Bob (yes/no) to interact with Seller, either:
   a) completing the payment and arranging delivery details
   b) canceling the transaction

# Example: A Two-Buyer Protocol


**Alice** and **Bob** cooperate in buying a book from **Seller** :

1. Alice sends a book title to Seller, who sends a quote back.
2. Alice checks whether Bob can contribute in buying the book.
3. Alice uses the answer from Bob (yes/no) to interact with Seller, either:
   a) completing the payment and arranging delivery details
   b) canceling the transaction
4. In case 3(a) Alice contacts Bob to get his address, and forwards it to Seller.

# Example: A Two-Buyer Protocol


**Alice** and **Bob** cooperate in buying a book from **Seller** :

1. Alice sends a book title to Seller, who sends a quote back.
2. Alice checks whether Bob can contribute in buying the book.
3. Alice uses the answer from Bob (yes/no) to interact with Seller, either:
   a) completing the payment and arranging delivery details
   b) canceling the transaction
4. In case 3(a) Alice contacts Bob to get his address, and forwards it to Seller.

4'. In case 3(b) Alice is in charge of gracefully concluding the conversation.

# Example: A Two-Buyer Protocol


**Alice** and **Bob** cooperate in buying a book from **Seller** :

1. Alice sends a book title to Seller, who sends a quote back.
2. Alice checks whether Bob can contribute in buying the book.
3. Alice uses the answer from Bob (yes/no) to interact with Seller, either:
   a) completing the payment and arranging delivery details
   b) canceling the transaction
4. In case 3(a) Alice contacts Bob to get his address, and forwards it to Seller.

4'. In case 3(b) Alice is in charge of gracefully concluding the conversation.

**Note:** The structure of messaging matters!

# Example: A Two-Buyer Protocol

Correctness goals for the implementations of Alice, Bob, and Seller:

- **Fidelity** – they **follow the intended protocol**.
  - Alice never ask Bob twice within the same conversation
  - Alice doesn't continue the transaction if Bob can't contribute
  - Alice chooses among the options provided by Seller

# Example: A Two-Buyer Protocol

Correctness goals for the implementations of Alice, Bob, and Seller:

- **Fidelity** – they **follow the intended protocol**.
- **Safety** – they don't feature **communication errors**.
  - Seller always returns an integer when Alice requests a quote

# Example: A Two-Buyer Protocol

Correctness goals for the implementations of Alice, Bob, and Seller:

- **Fidelity** – they **follow the intended protocol**.
- **Safety** – they don't feature **communication errors**.
- **Deadlock-Freedom** – they do not **"get stuck"** while running the protocol.
  - Alice eventually receives an answer from Bob on his contribution.

# Example: A Two-Buyer Protocol

Correctness goals for the implementations of Alice, Bob, and Seller:

- **Fidelity** – they **follow the intended protocol**.
- **Safety** – they don't feature **communication errors**.
- **Deadlock-Freedom** – they do not **"get stuck"** while running the protocol.
- **Termination** – they do not engage in **infinite behavior** (that may prevent them from completing the protocol)

# Example: A Two-Buyer Protocol

Correctness goals for the implementations of Alice, Bob, and Seller:

- **Fidelity** – they **follow the intended protocol**.
- **Safety** – they don't feature **communication errors**.
- **Deadlock-Freedom** – they do not **"get stuck"** while running the protocol.
- **Termination** – they do not engage in **infinite behavior** (that may prevent them from completing the protocol)

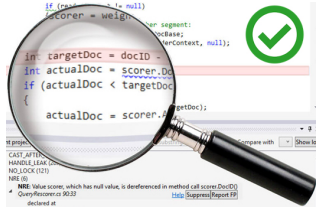
## Correctness

- A **non-trivial notion**, which follows from the interplay of these properties.
- **Hard to enforce** due to actions being "scattered around" in programs.

→ A session specifies a protocol's structure, enabling program verification. It stipulates **what** and **when** should be exchanged (along a channel)

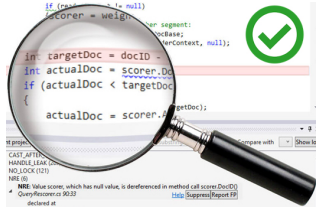
# Type Systems

- Can detect bugs before programs are run
- Attached to many programming languages
- Implement a specific notion of correctness
  A program is either correct or incorrect



# Type Systems

- Can detect bugs before programs are run
- Attached to many programming languages
- Implement a specific notion of correctness
  A program is either correct or incorrect

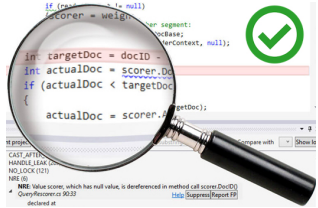



## Sequential Languages

- **Data** type systems classify values in a program
- Examples: Integers, strings, etc

# Type Systems

- Can detect bugs before programs are run
- Attached to many programming languages
- Implement a specific notion of correctness
  A program is either correct or incorrect




## Sequential Languages

- **Data** type systems classify values in a program
- Examples: Integers, strings, etc

## Concurrent Languages

- **Behavioral** type systems classify protocols in a program
- Example: “first send username, then receive true/false, finally close”
- A typical bug: sending messages in the wrong order

# How to Design Type Systems? Logic to the Rescue!

# How to Design Type Systems? Logic to the Rescue!

A landmark result in programming language theory:

- **Propositions as types** (Curry, 1935; Howard, 1969)

| | | |
|---|---|---|
| Propositions in Intuitionistic Logic | $\Longleftrightarrow$ | Types |
| Proofs in natural deduction | $\Longleftrightarrow$ | Sequential programs ($\lambda$-calculus) |
| Proof simplification | $\Longleftrightarrow$ | Program evaluation ($\beta$-reduction) |

# How to Design Type Systems? Logic to the Rescue!

- **Propositions as types** (Curry, 1935; Howard, 1969)

| | | |
|---|---|---|
| Propositions in Intuitionistic Logic | $\Longleftrightarrow$ | Types |
| Proofs in natural deduction | $\Longleftrightarrow$ | Sequential programs ($\lambda$-calculus) |
| Proof simplification | $\Longleftrightarrow$ | Program evaluation ($\beta$-reduction) |

- What about concurrent programs? A **resource-oriented** view!

# How to Design Type Systems? Logic to the Rescue!

- **Propositions as sessions**

| | | |
|---|---|---|
| Propositions in **Linear Logic** (LL) | $\leftrightarrow$ | **Session types** |
| Proofs in LL | $\leftrightarrow$ | Interacting processes |
| Cut elimination in LL | $\leftrightarrow$ | process communication |

# Outline

# Protocols as Session Types

Session types specify protocols in terms of

- communication actions: send and receive
- choices: offers and selections
- recursion

# Protocols as Session Types

Session types specify protocols in terms of

- communication actions: send and receive
- choices: offers and selections
- recursion

Session protocols are attached to **interaction devices**:

- communication channels in programs (think Go and Rust)
- TCP-IP sockets
- ...

# Protocols as Session Types

A formal syntax for protocols:

$S ::= \ !U; S$ **send value** of type $U$, continue as $S$

# Protocols as Session Types

A formal syntax for protocols:

$$
\begin{array}{rll}
S ::= & !U; S & \textbf{send value} \text{ of type } U, \text{ continue as } S \\
\mid & ?U; S & \textbf{receive value} \text{ of type } U, \text{ continue as } S
\end{array}
$$

# Protocols as Session Types

A formal syntax for protocols:

$$
\begin{array}{rll}
S ::= & !U;S & \textbf{send value} \text{ of type } U, \text{ continue as } S \\
\mid & ?U;S & \textbf{receive value} \text{ of type } U, \text{ continue as } S \\
\mid & \&\{l_1 : S_1, \cdots, l_n : S_n\} & \textbf{offer} \text{ alternatives } S_1, \ldots, S_n \\
 & & (l_1, \ldots, l_n \text{ are } \textbf{labels})
\end{array}
$$

# Protocols as Session Types

A formal syntax for protocols:

$$
\begin{array}{rll}
S ::= & !U;S & \textbf{send value} \text{ of type } U, \text{ continue as } S \\
\mid & ?U;S & \textbf{receive value} \text{ of type } U, \text{ continue as } S \\
\mid & \&\{l_1 : S_1, \cdots, l_n : S_n\} & \textbf{offer} \text{ alternatives } S_1, \ldots, S_n \\
 & & \quad (l_1, \ldots, l_n \text{ are } \textbf{labels}) \\
\mid & \oplus\{l_1 : S_1, \cdots, l_n : S_n\} & \textbf{select} \text{ one between } S_1, \ldots, S_n
\end{array}
$$

# Protocols as Session Types

A formal syntax for protocols:

$$
\begin{array}{rll}
S ::= & !U;S & \textbf{send value} \text{ of type } U, \text{ continue as } S \\
\mid & ?U;S & \textbf{receive value} \text{ of type } U, \text{ continue as } S \\
\mid & \&\{l_1 : S_1, \cdots, l_n : S_n\} & \textbf{offer} \text{ alternatives } S_1, \ldots, S_n \\
 & & \quad (l_1, \ldots, l_n \text{ are } \textbf{labels}) \\
\mid & \oplus\{l_1 : S_1, \cdots, l_n : S_n\} & \textbf{select} \text{ one between } S_1, \ldots, S_n \\
\mid & \mu t.S \mid t & \textbf{recursion}
\end{array}
$$

# Protocols as Session Types

A formal syntax for protocols:

$$
\begin{array}{rll}
S ::= & !U;S & \textbf{send value} \text{ of type } U, \text{ continue as } S \\
\mid & ?U;S & \textbf{receive value} \text{ of type } U, \text{ continue as } S \\
\mid & \&\{l_1 : S_1, \cdots, l_n : S_n\} & \textbf{offer} \text{ alternatives } S_1, \ldots, S_n \\
 & & \quad (l_1, \ldots, l_n \text{ are } \textbf{labels}) \\
\mid & \oplus\{l_1 : S_1, \cdots, l_n : S_n\} & \textbf{select} \text{ one between } S_1, \ldots, S_n \\
\mid & \mu t.S \mid t & \textbf{recursion} \\
\mid & \mathtt{end} & \textbf{terminated protocol}
\end{array}
$$

# Protocols as Session Types

A formal syntax for protocols:

$$
\begin{array}{lll}
S ::= & !U;S & \textbf{send value} \text{ of type } U, \text{ continue as } S \\
\mid & ?U;S & \textbf{receive value} \text{ of type } U, \text{ continue as } S \\
\mid & \&\{l_1 : S_1, \cdots, l_n : S_n\} & \textbf{offer} \text{ alternatives } S_1, \ldots, S_n \\
 & & \quad (l_1, \ldots, l_n \text{ are } \textbf{labels}) \\
\mid & \oplus\{l_1 : S_1, \cdots, l_n : S_n\} & \textbf{select} \text{ one between } S_1, \ldots, S_n \\
\mid & \mu t.S \mid t & \textbf{recursion} \\
\mid & \texttt{end} & \textbf{terminated protocol}
\end{array}
$$

**Notice**:

- Sequential communication patterns (no built-in concurrency).
- $U$ stands for basic values (e.g. `int`) but also other sessions $S$.

# Exercises

What do these protocols do?

- $S_1 \triangleq ?\texttt{int}; \texttt{end}$

# Exercises

What do these protocols do?

- $S_1 \triangleq ?\texttt{int}; \texttt{end}$
- $S_2 \triangleq ?\texttt{int}; ?\texttt{int}; !\texttt{bool}; \texttt{end}$
- $S_3 \triangleq !\texttt{int}; ?\texttt{bool}; \texttt{end}$

# Exercises

What do these protocols do?

- $S_1 \triangleq ?\texttt{int}; \texttt{end}$
- $S_2 \triangleq ?\texttt{int}; ?\texttt{int}; !\texttt{bool}; \texttt{end}$
- $S_3 \triangleq !\texttt{int}; ?\texttt{bool}; \texttt{end}$
- $S_4 \triangleq \&\{l_1 : ?\texttt{int}; ?\texttt{int}; !\texttt{bool}; \texttt{end}, \quad l_2 : ?\texttt{str}; !\texttt{int}; \texttt{end}\}$

# Exercises

What do these protocols do?

- $S_1 \triangleq ?\texttt{int}; \texttt{end}$
- $S_2 \triangleq ?\texttt{int}; ?\texttt{int}; !\texttt{bool}; \texttt{end}$
- $S_3 \triangleq !\texttt{int}; ?\texttt{bool}; \texttt{end}$
- $S_4 \triangleq \&\{l_1 : ?\texttt{int}; ?\texttt{int}; !\texttt{bool}; \texttt{end}, \quad l_2 : ?\texttt{str}; !\texttt{int}; \texttt{end}\}$
- $S_5 \triangleq \mu t.\, ?\texttt{int}; t$
- $S_6 \triangleq \mu t.\, ?\texttt{int}; ?\texttt{int}; !\texttt{bool}; t$

# Exercises

What do these protocols do?

- $S_1 \triangleq ?\texttt{int}; \texttt{end}$
- $S_2 \triangleq ?\texttt{int}; ?\texttt{int}; !\texttt{bool}; \texttt{end}$
- $S_3 \triangleq !\texttt{int}; ?\texttt{bool}; \texttt{end}$
- $S_4 \triangleq \&\{l_1 : ?\texttt{int}; ?\texttt{int}; !\texttt{bool}; \texttt{end}, \quad l_2 : ?\texttt{str}; !\texttt{int}; \texttt{end}\}$
- $S_5 \triangleq \mu t.\, ?\texttt{int}; t$
- $S_6 \triangleq \mu t.\, ?\texttt{int}; ?\texttt{int}; !\texttt{bool}; t$
- $S_7 \triangleq \mu t.\, \& \{\mathit{recv} : ?\texttt{int}; ?\texttt{int}; !\texttt{bool}; t, \quad \mathit{stop} : !\texttt{str}; \texttt{end}\}$

# Exercises

What do these protocols do?

- $S_1 \triangleq ?\texttt{int}; \texttt{end}$
- $S_2 \triangleq ?\texttt{int}; ?\texttt{int}; !\texttt{bool}; \texttt{end}$
- $S_3 \triangleq !\texttt{int}; ?\texttt{bool}; \texttt{end}$
- $S_4 \triangleq \&\{l_1 : ?\texttt{int}; ?\texttt{int}; !\texttt{bool}; \texttt{end}, \quad l_2 : ?\texttt{str}; !\texttt{int}; \texttt{end}\}$
- $S_5 \triangleq \mu t.\, ?\texttt{int}; t$
- $S_6 \triangleq \mu t.\, ?\texttt{int}; ?\texttt{int}; !\texttt{bool}; t$
- $S_7 \triangleq \mu t.\ \& \{\mathit{recv} : ?\texttt{int}; ?\texttt{int}; !\texttt{bool}; t, \quad \mathit{stop} : !\texttt{str}; \texttt{end}\}$
- $S_8 \triangleq \oplus\{l_1 : !\texttt{int}; !\texttt{int}; ?\texttt{bool}; \texttt{end}, \quad l_2 : !\texttt{str}; ?\texttt{int}; \texttt{end}\}$

# The Two-Buyer Protocol, Revisited

Rakuten kobo

Recall the protocol between Alice, Bob, and Seller:

1. Alice sends a book title to Seller, who sends a quote back.
2. Alice checks whether Bob can contribute to buy the book.
3. Alice uses the answer from Bob (yes/no) to interact with Seller, either:
   a) completing the payment and arranging delivery details
   b) canceling the transaction
4. In case 3(a) Alice contacts Bob to get his address, and forwards it to Seller.

4'. In case 3(b) Alice is in charge of gracefully concluding the conversation.

# Session Types for The Two-Buyer Protocol

Two independent protocols, with Alice "leading" the interactions:

1. A session type for Seller (in its interaction with Alice):

$$S_{\mathsf{SA}} = ?\texttt{book};\ !\texttt{quote};\ \&\begin{cases} \text{buy}: & ?\texttt{paym};\ ?\texttt{address};\ !\texttt{ok};\ \text{end} \\ \text{cancel}: & ?\texttt{thanks};\ !\texttt{bye};\ \text{end} \end{cases}$$

# Session Types for The Two-Buyer Protocol

Two independent protocols, with Alice "leading" the interactions:

1. A session type for Seller (in its interaction with Alice):

$$S_{\mathsf{SA}} = \texttt{?book}; \texttt{!quote}; \& \begin{cases} \text{buy}: & \texttt{?paym}; \texttt{?address}; \texttt{!ok}; \text{end} \\ \text{cancel}: & \texttt{?thanks}; \texttt{!bye}; \text{end} \end{cases}$$

2. A session type for Alice (in its interaction with Bob):

$$S_{\mathsf{AB}} = \texttt{!cost}; \& \begin{cases} \text{share}: & \texttt{?address}; \texttt{!ok}; \text{end} \\ \text{close}: & \texttt{!bye}; \text{end} \end{cases}$$

# Session Types for The Two-Buyer Protocol

Implementations for Alice, Bob, and Seller should be **compatible**.

# Session Types for The Two-Buyer Protocol

Implementations for Alice, Bob, and Seller should be **compatible**.

**Duality** relates session types with opposite behaviors:

- The dual of sending is receiving (and vice versa)
- Branching is the dual of selection (and vice versa)

The dual of session type $S$ is written $\overline{S}$.

# Session Types for The Two-Buyer Protocol

**Example**:

- Recall that $S_{\mathsf{AB}}$ describes Alice's viewpoint in her interaction with Bob:

$$S_{\mathsf{AB}} = \; !\texttt{cost}; \& \begin{cases} \text{share}: & ?\texttt{address}; \; !\texttt{ok}; \; \text{end} \\ \text{close}: & !\texttt{bye}; \; \text{end} \end{cases}$$

- Given this, Bob's implementation should conform to $\overline{S_{\mathsf{AB}}}$, the dual of $S_{\mathsf{AB}}$:

# Session Types for The Two-Buyer Protocol

**Example**:

- Recall that $S_{\mathsf{AB}}$ describes Alice's viewpoint in her interaction with Bob:

$$S_{\mathsf{AB}} = \ !\texttt{cost}; \& \begin{cases} \text{share}: & ?\texttt{address}; \ !\texttt{ok}; \ \text{end} \\ \text{close}: & !\texttt{bye}; \ \text{end} \end{cases}$$

- Given this, Bob's implementation should conform to $\overline{S_{\mathsf{AB}}}$, the dual of $S_{\mathsf{AB}}$:

$$\overline{S_{\mathsf{AB}}} = \ ?\texttt{cost}; \oplus \begin{cases} \text{share}: & !\texttt{address}; \ ?\texttt{ok}; \ \text{end} \\ \text{close}: & ?\texttt{bye}; \ \text{end} \end{cases}$$

# Session Types for The Two-Buyer Protocol

**Example**:

- Recall that $S_{\mathsf{AB}}$ describes Alice's viewpoint in her interaction with Bob:

$$S_{\mathsf{AB}} = \ !\texttt{cost}; \& \begin{cases} \text{share}: & ?\texttt{address}; \ !\texttt{ok}; \ \text{end} \\ \text{close}: & !\texttt{bye}; \ \text{end} \end{cases}$$

- Given this, Bob's implementation should conform to $\overline{S_{\mathsf{AB}}}$, the dual of $S_{\mathsf{AB}}$:

$$\overline{S_{\mathsf{AB}}} = \ ?\texttt{cost}; \oplus \begin{cases} \text{share}: & !\texttt{address}; \ ?\texttt{ok}; \ \text{end} \\ \text{close}: & ?\texttt{bye}; \ \text{end} \end{cases}$$

- Also, Alice's implementation should conform to both $\overline{S_{\mathsf{SA}}}$ and $S_{\mathsf{AB}}$.

# Session Type Duality, Formally

Given a (finite) session type $S$, its dual type $\overline{S}$ is inductively defined as follows:

$$\begin{aligned}
\overline{!U;S} &= ?U;\overline{S} \\
\overline{?U;S} &= !U;\overline{S} \\
\overline{\&\{l_i : S_i\}_{i\in I}} &= \oplus\{l_i : \overline{S_i}\}_{i\in I} \\
\overline{\oplus\{l_i : S_i\}_{i\in I}} &= \&\{l_i : \overline{S_i}\}_{i\in I} \\
\overline{\mathsf{end}} &= \mathsf{end}
\end{aligned}$$

# Session Type Duality, Formally

Given a (finite) session type $S$, its dual type $\overline{S}$ is inductively defined as follows:

$$\begin{aligned}
\overline{!U;S} &= ?U;\overline{S} \\
\overline{?U;S} &= !U;\overline{S} \\
\overline{\&\{l_i : S_i\}_{i\in I}} &= \oplus\{l_i : \overline{S_i}\}_{i\in I} \\
\overline{\oplus\{l_i : S_i\}_{i\in I}} &= \&\{l_i : \overline{S_i}\}_{i\in I} \\
\overline{\mathsf{end}} &= \mathsf{end}
\end{aligned}$$

**Notice**:

- For recursive types, duality is defined **coinductively** (the dual of $\mu t.S$ is *not* $\mu t.\overline{S}$)

# Many Classes of Session Types

There are many classes/variants of session types:

- We overviewed **binary** sessions, where types describe two-party protocols. (We used two separate types to handle Alice, Bob, and Seller.)
- With **multiparty** sessions, types describe protocols involving more than two participants. There is a single **global type** and multiple **local types**.

# Many Classes of Session Types (contd.)

There are many classes/variants of session types:

- We saw a syntax of types that describes **regular behaviors**:
  in '$?U; S$' and '$!U; S$' we use ';' to compose an **action** and a **protocol**.
- In **context-free** session types, we can freely compose two protocols: '$S_1; S_2$'.
  This is arguably the most expressive class of protocols (to date).

# Many Classes of Session Types (contd.)

There are many classes/variants of session types:

- Session types can be attached to programming languages with **synchronous** and **asynchronous** communication.
  - Synchronous: A message is received as soon as it is sent
  - Asynchronous: Message reception is not immediate

  These operational differences are reflected in types.
  The real world is asynchronous but synchronous is easier to define.

# Many Classes of Session Types (contd.)

There are many classes/variants of session types:

- Session types can be attached to programming languages with **synchronous** and **asynchronous** communication.
  - Synchronous: A message is received as soon as it is sent
  - Asynchronous: Message reception is not immediate

  These operational differences are reflected in types.
  The real world is asynchronous but synchronous is easier to define.

- Session types are **paradigm-independent**.
  They have been adapted to functional, object-oriented, and actor-based languages with concurrency.

# Taking Stock

Up to here:

- Correctness for communicating programs
- Sessions as protocol specifications
- A formal syntax for session types
- Example: A two-buyer protocol
- Protocol compatibility via duality for session types

**Coming next**:

- Linear logic, in its intuitionistic variant

# Linear Logic

# Outline

# Intuitionistic Logic (IL)

The grammar of propositions:

$$A, B \quad ::= \quad X \mid A \to B \mid A \times B \mid A + B$$

where:

- $X$ is a propositional **constant**
- $A \to B$ denotes **implication**
- $A \times B$ denotes **conjunction** (alternative notation for $A \wedge B$)
- $A + B$ denotes **disjunction** (alternative notation for $A \vee B$)

# Intuitionistic Logic (IL)

The grammar of propositions:

$$A, B \;::=\; X \mid A \to B \mid A \times B \mid A + B$$

where:

- $X$ is a propositional **constant**
- $A \to B$ denotes **implication**
- $A \times B$ denotes **conjunction** (alternative notation for $A \wedge B$)
- $A + B$ denotes **disjunction** (alternative notation for $A \vee B$)

Assumptions and judgments:

- $\Gamma, \Delta$ denote **assumptions**: sequences of zero or more propositions.
- A **judgment** $\Gamma \vdash A$ means "from assumptions $\Gamma$ we can conclude $A$".

# Rules for Intuitionistic Logic (IL)

Proving judgments of the form $\Gamma \vdash A$.

- A single **axiom**, a rule without premises.

# Rules for Intuitionistic Logic (IL)

Proving judgments of the form $\Gamma \vdash A$.

- A single **axiom**, a rule without premises.
- **Structural rules** control the ordering, duplication, and discarding of hypotheses in the assumption $\Gamma$.

# Rules for Intuitionistic Logic (IL)

Proving judgments of the form $\Gamma \vdash A$.

- A single **axiom**, a rule without premises.
- **Structural rules** control the ordering, duplication, and discarding of hypotheses in the assumption $\Gamma$.
- Each logical connective is specified in terms of **two rules**:
  an introduction rule and an elimination rule (this is called **natural deduction**).

# Rules for Intuitionistic Logic (IL)

Proving judgments of the form $\Gamma \vdash A$.

- A single **axiom**, a rule without premises.
- **Structural rules** control the ordering, duplication, and discarding of hypotheses in the assumption $\Gamma$.
- Each logical connective is specified in terms of **two rules**:
  an introduction rule and an elimination rule (this is called **natural deduction**).
- Using the rules, we aim to construct **derivations** for judgements.

# Rules for Intuitionistic Logic (IL)

Proving judgments of the form $\Gamma \vdash A$.

- A single **axiom**, a rule without premises.
- **Structural rules** control the ordering, duplication, and discarding of hypotheses in the assumption $\Gamma$.
- Each logical connective is specified in terms of **two rules**:
  an introduction rule and an elimination rule (this is called **natural deduction**).
- Using the rules, we aim to construct **derivations** for judgements.

(Later we will see rules in a **sequent calculus**, rather than in natural deduction.)

# Rules for IL

$$\text{Id}\ \frac{}{A \vdash A} \qquad \text{Exchange}\ \frac{\Gamma, \Delta \vdash A}{\Delta, \Gamma \vdash A} \qquad \text{Contraction}\ \frac{\Gamma, A, A \vdash B}{\Gamma, A \vdash B} \qquad \text{Weakening}\ \frac{\Gamma \vdash B}{\Gamma, A \vdash B}$$

$$\to\text{-I}\ \frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B} \qquad \to\text{-E}\ \frac{\Gamma \vdash A \to B \qquad \Delta \vdash A}{\Gamma, \Delta \vdash B}$$

$$\times\text{-I}\ \frac{\Gamma \vdash A \qquad \Delta \vdash B}{\Gamma, \Delta \vdash A \times B} \qquad \times\text{-E}\ \frac{\Gamma \vdash A \times B \qquad \Delta, A, B \vdash C}{\Gamma, \Delta \vdash C}$$

$$+\text{-I}_1\ \frac{\Gamma \vdash A}{\Gamma \vdash A + B} \qquad +\text{-I}_2\ \frac{\Gamma \vdash B}{\Gamma \vdash A + B} \qquad +\text{-E}\ \frac{\Gamma \vdash A + B \qquad \Delta, A \vdash C \qquad \Delta, B \vdash C}{\Gamma, \Delta \vdash C}$$

# Example of a Derivation

A derivation for the judgment

$$A \to B, A \vdash A \times B$$

expressed as a **proof tree** (labels for the rules are omitted):

$$\dfrac{\dfrac{\dfrac{}{A \vdash A} \qquad \dfrac{\dfrac{}{A \to B \vdash A \to B} \qquad \dfrac{}{A \vdash A}}{A \to B, A \vdash B}}{A, A \to B, A \vdash A \times B}}{\dfrac{A \to B, A, A \vdash A \times B}{A \to B, A \vdash A \times B}}$$

We see how $A$ is used twice, with the help of contraction and exchange rules.

# Alternative Rules

We have just seen:

$$\times\text{-I}\quad \frac{\Gamma \vdash A \qquad \Delta \vdash B}{\Gamma, \Delta \vdash A \times B} \qquad\qquad \times\text{-E}\quad \frac{\Gamma \vdash A \times B \qquad \Delta, A, B \vdash C}{\Gamma, \Delta \vdash C}$$

We can also have:

$$\times\text{-I'}\quad \frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \times B} \qquad \times\text{-E}_1\quad \frac{\Gamma \vdash A \times B}{\Gamma \vdash A} \qquad \times\text{-E}_2\quad \frac{\Gamma \vdash A \times B}{\Gamma \vdash B}$$

The rules are **equivalent**: they are interderivable using structural rules.

# More Alternative Rules

We have just seen:

$$\text{Id}\ \frac{}{A \vdash A} \qquad \to\text{-I}\ \frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B} \qquad \to\text{-E}\ \frac{\Gamma \vdash A \to B \qquad \Delta \vdash A}{\Gamma, \Delta \vdash B}$$

We can also have:

$$\text{Id'}\ \frac{}{\Gamma, A, \Delta \vdash A} \qquad \to\text{-I}\ \frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B} \qquad \to\text{-E'}\ \frac{\Gamma \vdash A \to B \qquad \Gamma \vdash A}{\Gamma \vdash B}$$

Here again, structural rules ensure that either group of rules can be used.
We prefer to have an explicit treatment of structural rules.

# Commuting Conversions

The order in which certain (structural) rules are applied is irrelevant.

An equivalence relation on proofs, denoted $\Longleftrightarrow$, via **commuting conversions**.

Here are the commuting conversions involving contraction and $\rightarrow$-E:

$$\dfrac{\dfrac{\Gamma, C, \Delta, C \vdash B}{\Gamma, C, \Delta \vdash B}}{\Gamma, C, \Delta \vdash B} \quad \Longleftrightarrow \quad \dfrac{\dfrac{\Gamma, C, \Delta, C \vdash B}{\Gamma, C, \Delta \vdash B}}{\Gamma, C, \Delta \vdash B}$$

$$\dfrac{\dfrac{\Gamma, \Delta, C, C \vdash B}{\Gamma, \Delta, C \vdash B}}{\Gamma, \Delta, C \vdash B} \quad \Longleftrightarrow \quad \dfrac{\dfrac{\Gamma, \Delta, C, C \vdash B}{\Gamma, \Delta, C \vdash B}}{\Gamma, \Delta, C \vdash B}$$

# Commuting Conversions

The order in which certain (structural) rules are applied is irrelevant.

An equivalence relation on proofs, denoted $\Longleftrightarrow$, via **commuting conversions**.

Here are the commuting conversions involving contraction and $\to$-E:

$$\dfrac{\dfrac{\Gamma, C, C \vdash A \to B}{\Gamma, C \vdash A \to B} \qquad \Delta \vdash A}{\Gamma, C, \Delta \vdash B} \quad \Longleftrightarrow \quad \dfrac{\dfrac{\Gamma, C, C \vdash A \to B \qquad \Delta \vdash A}{\Gamma, C, C, \Delta \vdash B}}{\Gamma, C, \Delta \vdash B}$$

$$\dfrac{}{\Gamma, \Delta, C \vdash B} \quad \Longleftrightarrow \quad \dfrac{}{\Gamma, \Delta, C \vdash B}$$

# Commuting Conversions

The order in which certain (structural) rules are applied is irrelevant.

An equivalence relation on proofs, denoted $\Longleftrightarrow$, via **commuting conversions**.

Here are the commuting conversions involving contraction and $\to$-E:

$$\dfrac{\dfrac{\Gamma, C, C \vdash A \to B}{\Gamma, C \vdash A \to B} \qquad \Delta \vdash A}{\Gamma, C, \Delta \vdash B} \quad \Longleftrightarrow \quad \dfrac{\Gamma, C, C \vdash A \to B \qquad \Delta \vdash A}{\dfrac{\Gamma, C, C, \Delta \vdash B}{\Gamma, C, \Delta \vdash B}}$$

$$\dfrac{\Gamma \vdash A \to B \qquad \dfrac{\Delta, C, C \vdash A}{\Delta, C \vdash A}}{\Gamma, \Delta, C \vdash B} \quad \Longleftrightarrow \quad \dfrac{\Gamma \vdash A \to B \qquad \Delta, C, C \vdash A}{\dfrac{\Gamma, \Delta, C, C \vdash B}{\Gamma, \Delta, C \vdash B}}$$

# Proof Reduction

- A judgement may have several, very different derivations.

# Proof Reduction

- A judgement may have several, very different derivations.
- In particular, it is not very useful to have an introduction rule for a connective that is followed by the corresponding elimination rule.

# Proof Reduction

- A judgement may have several, very different derivations.
- In particular, it is not very useful to have an introduction rule for a connective that is followed by the corresponding elimination rule.
- The interest is in **reducing** proofs to avoid unnecessary occurrence of connectives; the proofs themselves may not be “smaller”.

# Proof Reduction

- A judgement may have several, very different derivations.
- In particular, it is not very useful to have an introduction rule for a connective that is followed by the corresponding elimination rule.
- The interest is in **reducing** proofs to avoid unnecessary occurrence of connectives; the proofs themselves may not be "smaller".
- Notions of **normal form** and **subformula property** apply for proofs and their reductions.

# Proof Reduction: Example

Consider the proof

$$\dfrac{\dfrac{\dfrac{\dfrac{}{B \vdash B} \quad \dfrac{}{B \vdash B}}{B, B \vdash B \times B}}{B \vdash B \times B}}{\vdash B \to (B \times B)} (\dagger) \quad \dfrac{\dfrac{}{A \to B \vdash A \to B} \quad \dfrac{}{A \vdash A}}{A \to B, A \vdash B}}{A \to B, A \vdash B \times B}$$

It can be reduced to:

$$\dfrac{\dfrac{\dfrac{}{A \to B \vdash A \to B} \quad \dfrac{}{A \vdash A}}{A \to B, A \vdash B} \quad \dfrac{\dfrac{}{A \to B \vdash A \to B} \quad \dfrac{}{A \vdash A}}{A \to B, A \vdash B}}{A \to B, A, A \to B, A \vdash B \times B}}{A \to B, A \vdash B \times B} (\dagger)$$

# Propositions as Types

Extending judgements to contain terms (= programs!).

- From the logic perspective, terms encode proofs.
- From the programmer's perspective, terms are a programming language for which logic provides a **type system**.

# Propositions as Types

Propositions can be read as types, and proofs are read as corresponding typed terms:

- A proof of $A \to B$ is a **function** that takes any proof of $A$ into a proof of $B$.
- A proof of $A \times B$ is a **pair** of a proof of $A$ and a proof of $B$.
- A proof of $A + B$ is a **disjoint sum**: either a proof of $A$ or a proof of $B$.

**Terms** encode the rules of the logic:

- An introduction rule: a term that **constructs** a value.
- An elimination rule: a term that **deconstructs** a value.
- There are no term forms for the structural rules.

We briefly have a look at the terms ($\lambda$-calculus!) and their reductions.

# Intuitionistic Terms

$$
\begin{aligned}
s, t, u, v, w \quad ::= \quad & x \\
& \mid \lambda x.\, u \mid s\,(t) \\
& \mid (t, u) \mid \mathtt{case}\; s\; \mathtt{of}\; (x, y) \to v \\
& \mid \mathtt{inl}\,(t) \mid \mathtt{inr}\,(u) \mid \mathtt{case}\; s\; \mathtt{of}\; \mathtt{inl}\,(x) \to v;\; \mathtt{inr}\,(y) \to w
\end{aligned}
$$

Assumptions $\Gamma, \Delta$ are now written in the form

$$x_1 : A_1, \ldots, x_n : A_n$$

where $n \geq 0$ and

- $x_1, \ldots, x_n$ are distinct variables;
- $A_1, \ldots, A_n$ are types (= propositions).

Judgments are then of the form $\Gamma \vdash t : A$ ("term $t$ has type $A$").

# Types for Intuitionistic Terms

Id

$$\frac{}{x : A \vdash x : A}$$

Exchange

$$\frac{\Gamma, \Delta \vdash t : A}{\Delta, \Gamma \vdash t : A}$$

Contraction

$$\frac{\Gamma, y : A, z : A \vdash u : B}{\Gamma, x : A \vdash u[x/y, x/z] : B}$$

Weakening

$$\frac{\Gamma \vdash u : B}{\Gamma, x : A \vdash u : B}$$

$\to$-I

$$\frac{\Gamma, x : A \vdash u : B}{\Gamma \vdash \lambda x.\, u : A \to B}$$

$\to$-E

$$\frac{\Gamma \vdash s : A \to B \qquad \Delta \vdash t : A}{\Gamma, \Delta \vdash s\,(t) : B}$$

$\times$-I

$$\frac{\Gamma \vdash t : A \qquad \Delta \vdash u : B}{\Gamma, \Delta \vdash (t, u) : A \times B}$$

$\times$-E

$$\frac{\Gamma \vdash s : A \times B \qquad \Delta, x : A, y : B \vdash v : C}{\Gamma, \Delta \vdash \mathtt{case}\ s\ \mathtt{of}\ (x, y) \to v : C}$$

# Types for Intuitionistic Terms

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash \mathtt{inl}\,(t) : A + B}\ +\text{-I}_1 \qquad \frac{\Gamma \vdash u : B}{\Gamma \vdash \mathtt{inr}\,(u) : A + B}\ +\text{-I}_2$$

$$\frac{\Gamma \vdash s : A + B \qquad \Delta, x : A \vdash v : C \qquad \Delta, y : B \vdash w : C}{\Gamma, \Delta \vdash \mathtt{case}\ s\ \mathtt{of}\ \mathtt{inl}\,(x) \to v;\ \mathtt{inr}\,(y) \to w : C}\ +\text{-E}$$

# Typing Derivations: Example

Recall the derivation for the judgment $A \to B, A \vdash A \times B$, seen before.
Now with terms:

$$\dfrac{\dfrac{}{y : A \vdash y : A} \qquad \dfrac{\dfrac{}{f : A \to B \vdash f : A \to B} \qquad \dfrac{}{z : A \vdash z : A}}{f : A \to B, z : A \vdash f\,(z) : B}}{\dfrac{y : A, f : A \to B, z : A \vdash (y, f\,(z)) : A \times B}{\dfrac{f : A \to B, y : A, z : A \vdash (y, f\,(z)) : A \times B}{f : A \to B, x : A \vdash (x, f\,(x)) : A \times B}}}$$

# Term Reduction

Since terms encode proofs, proof reductions in logic correspond to term reductions:

$$\lambda x.\, u\,(t) \Longleftrightarrow u[t/x]$$
$$\mathtt{case}\,(t, u)\,\mathtt{of}\,(x, y) \to v \Longleftrightarrow u[t/x, v/y]$$
$$\mathtt{case}\,\mathtt{inl}\,(t)\,\mathtt{of}\;\mathtt{inl}\,(x) \to v;\;\mathtt{inr}\,(y) \to w \Longleftrightarrow v[t/x]$$
$$\mathtt{case}\,\mathtt{inr}\,(u)\,\mathtt{of}\;\mathtt{inl}\,(x) \to v;\;\mathtt{inr}\,(y) \to w \Longleftrightarrow w[u/x]$$

**Notice:**
The notion of proof reduction, associated to properties of proof systems, corresponds precisely to term reduction for a model of computation!

# Outline

# Linear Logic: A Substructural Logic for Computation

- Introduced by Jean-Yves Girard in 1987.
- **Observation**: Contraction arbitrarily duplicates assumptions; weakening freely discards them. Not always sensible for computational resources!

# Linear Logic: A Substructural Logic for Computation

- Introduced by Jean-Yves Girard in 1987.
- **Observation**: Contraction arbitrarily duplicates assumptions; weakening freely discards them. Not always sensible for computational resources!
- Linear logic advocates a **disciplined** use of resources, by controlling contraction and weakening. Many applications in proof theory and computer science.
- A "resource-aware" logic:
  - $A \vdash B$: given $A$-resources, a way to obtain $B$-resources
  - Proof theoretically: **substructural logic**

# Linear Logic

- We are mostly interested in the intuitionistic variant, dubbed ILL.
  We will also have a look at classical LL (CLL).
- We see rules as a sequent calculus, rather than as natural deduction.
  - In natural deduction: introduction and elimination rules
  - In sequent calculi: left and right rules, plus the cut rule

# Linear Logic: Propositions

'$A$ is true' vs 'I have $A$-resources'

| | |
|---|---|
| $A \to B$ | if $A$ is true, then $B$ is true |
| $A \wedge B$ | both $A$ and $B$ are true |

# Linear Logic: Propositions

'$A$ is true' vs 'I have $A$-resources'

| | |
|---|---|
| $A \to B$ | if $A$ is true, then $B$ is true |
| $A \wedge B$ | both $A$ and $B$ are true |
| $A \multimap B$ | if you give me $A$, then I can produce $B$ |

# Linear Logic: Propositions

'$A$ is true' vs 'I have $A$-resources'

| | |
|---|---|
| $A \to B$ | if $A$ is true, then $B$ is true |
| $A \wedge B$ | both $A$ and $B$ are true |
| $A \multimap B$ | if you give me $A$, then I can produce $B$ |
| $A \otimes B$ | I have both $A$ and $B$ |

# Linear Logic: Propositions

'$A$ is true' vs 'I have $A$-resources'

| | |
|---|---|
| $A \to B$ | if $A$ is true, then $B$ is true |
| $A \wedge B$ | both $A$ and $B$ are true |

| | |
|---|---|
| $A \multimap B$ | if you give me $A$, then I can produce $B$ |
| $A \otimes B$ | I have both $A$ and $B$ |
| $A \,\&\, B$ | I can give you either $A$ or $B$ (you control the choice) |
| $A \oplus B$ | I choose to give you either $A$ or $B$ (I control the choice) |

**Examples**:

- $euro \otimes euro \multimap pizza$
- $dough \otimes (sauce \otimes toppings) \multimap pizza$

# Linear Logic: Linearity

$$A \to A \wedge A \qquad A \not\vdash A \otimes A$$

$$A \wedge B \to A \qquad A \otimes B \not\vdash A$$

# Linear Logic: Linearity

$$A \to A \wedge A \qquad A \not\vdash A \otimes A$$

$$A \wedge B \to A \qquad A \otimes B \not\vdash A$$

$$A \wedge (A \to B) \to B \qquad A \otimes (A \multimap B) \multimap B$$

# Linear Logic: Linearity

$$A \to A \wedge A \qquad A \not\multimap A \otimes A$$

$$A \wedge B \to A \qquad A \otimes B \not\multimap A$$

$$A \wedge (A \to B) \to B \qquad A \otimes (A \multimap B) \multimap B$$

$$A \wedge (A \to B) \to A \wedge B \qquad A \otimes (A \multimap B) \not\multimap A \otimes B$$

Linear Logic is **substructural**, resource-aware.

# Outline

# Proof Theory of $\wedge$

Sequent calculus: $\Gamma \vdash A$

$$\frac{}{A \vdash A} \qquad \wedge\mathrm{R}\ \frac{\Gamma_1 \vdash A \qquad \Gamma_2 \vdash B}{\Gamma_1, \Gamma_2 \vdash A \wedge B} \qquad \wedge\mathrm{L}\ \frac{\Gamma, A, B \vdash C}{\Gamma, A \wedge B \vdash C}$$

$$\mathrm{Weakn}\ \frac{\Gamma \vdash C}{\Gamma, A \vdash C} \qquad \mathrm{Contr}\ \frac{\Gamma, A, A \vdash C}{\Gamma, A \vdash C} \qquad \mathrm{eXchg}\ \frac{\Gamma, A, B, \Gamma' \vdash C}{\Gamma, B, A, \Gamma' \vdash C}$$

# Proof Theory of $\wedge$

Assertion $A$ holds under the list of hypotheses $\Gamma$.
Intuitively, $\bigwedge \Gamma \implies A$

Sequent calculus: $\Gamma \vdash A$

$$\frac{}{A \vdash A}$$

$$\wedge\mathrm{R}\ \frac{\Gamma_1 \vdash A \qquad \Gamma_2 \vdash B}{\Gamma_1, \Gamma_2 \vdash A \wedge B}$$

$$\wedge\mathrm{L}\ \frac{\Gamma, A, B \vdash C}{\Gamma, A \wedge B \vdash C}$$

$$\mathrm{Weakn}\ \frac{\Gamma \vdash C}{\Gamma, A \vdash C}$$

$$\mathrm{Contr}\ \frac{\Gamma, A, A \vdash C}{\Gamma, A \vdash C}$$

$$\mathrm{eXchg}\ \frac{\Gamma, A, B, \Gamma' \vdash C}{\Gamma, B, A, \Gamma' \vdash C}$$

# Proof Theory of $\wedge$

Axiom rule

Sequent calculus: $\Gamma \vdash A$

$$\frac{}{A \vdash A}$$

$$\wedge\mathrm{R}\ \frac{\Gamma_1 \vdash A \qquad \Gamma_2 \vdash B}{\Gamma_1, \Gamma_2 \vdash A \wedge B}$$

$$\wedge\mathrm{L}\ \frac{\Gamma, A, B \vdash C}{\Gamma, A \wedge B \vdash C}$$

$$\mathrm{Weakn}\ \frac{\Gamma \vdash C}{\Gamma, A \vdash C}$$

$$\mathrm{Contr}\ \frac{\Gamma, A, A \vdash C}{\Gamma, A \vdash C}$$

$$\mathrm{eXchg}\ \frac{\Gamma, A, B, \Gamma' \vdash C}{\Gamma, B, A, \Gamma' \vdash C}$$

# Proof Theory of $\wedge$

Sequent calculus: $\Gamma \vdash A$

Right and left rules for $\wedge$

$$\frac{}{A \vdash A}$$

$$\wedge\mathrm{R}\ \frac{\Gamma_1 \vdash A \qquad \Gamma_2 \vdash B}{\Gamma_1, \Gamma_2 \vdash A \wedge B}$$

$$\wedge\mathrm{L}\ \frac{\Gamma, A, B \vdash C}{\Gamma, A \wedge B \vdash C}$$

$$\mathrm{Weakn}\ \frac{\Gamma \vdash C}{\Gamma, A \vdash C}$$

$$\mathrm{Contr}\ \frac{\Gamma, A, A \vdash C}{\Gamma, A \vdash C}$$

$$\mathrm{eXchg}\ \frac{\Gamma, A, B, \Gamma' \vdash C}{\Gamma, B, A, \Gamma' \vdash C}$$

# Proof Theory of $\wedge$

Sequent calculus: $\Gamma \vdash A$

$$\frac{}{A \vdash A} \qquad \wedge\mathrm{R}\ \frac{\Gamma_1 \vdash A \qquad \Gamma_2 \vdash B}{\Gamma_1, \Gamma_2 \vdash A \wedge B} \qquad \wedge\mathrm{L}\ \frac{\Gamma, A, B \vdash C}{\Gamma, A \wedge B \vdash C}$$

$$\mathrm{Weakn}\ \frac{\Gamma \vdash C}{\Gamma, A \vdash C} \qquad \mathrm{Contr}\ \frac{\Gamma, A, A \vdash C}{\Gamma, A \vdash C} \qquad \mathrm{eXchg}\ \frac{\Gamma, A, B, \Gamma' \vdash C}{\Gamma, B, A, \Gamma' \vdash C}$$

Structural rules

# Proof Theory of $\wedge$

$\Gamma : \mathit{Frml} \to \mathbb{N}$ is a multiset

Sequent calculus: $\Gamma \vdash A$

$$\frac{}{A \vdash A}$$

$$\wedge\mathrm{R}\ \frac{\Gamma_1 \vdash A \qquad \Gamma_2 \vdash B}{\Gamma_1, \Gamma_2 \vdash A \wedge B}$$

$$\wedge\mathrm{L}\ \frac{\Gamma, A, B \vdash C}{\Gamma, A \wedge B \vdash C}$$

$$\mathrm{Weakn}\ \frac{\Gamma \vdash C}{\Gamma, A \vdash C}$$

$$\mathrm{Contr}\ \frac{\Gamma, A, A \vdash C}{\Gamma, A \vdash C}$$

$$\mathrm{eXchg}\ \frac{\Gamma, A, B, \Gamma' \vdash C}{\Gamma, B, A, \Gamma' \vdash C}$$

We can omit the exchange rule by treating $\Gamma$ as a multiset

# Generalized Structural Rules

$$\frac{\Gamma, \Gamma \vdash C}{\Gamma \vdash C} \qquad \frac{\Gamma \vdash C}{\Gamma, \Delta \vdash C}$$

Obtained from repeatedly using the usual rules (and exchange).

# Using Structural Rules

$$\frac{}{A \wedge B \vdash A} \qquad \frac{}{A \vdash A \wedge A}$$

# Using Structural Rules

$$\dfrac{\dfrac{}{A, B \vdash A}}{A \wedge B \vdash A} \qquad \dfrac{}{A \vdash A \wedge A}$$

# Using Structural Rules

$$\dfrac{\dfrac{A \vdash A}{A, B \vdash A}}{A \wedge B \vdash A} \qquad \dfrac{}{A \vdash A \wedge A}$$

# Using Structural Rules

$$\dfrac{\dfrac{A \vdash A}{A, B \vdash A}}{A \wedge B \vdash A} \qquad \dfrac{\dfrac{}{A, A \vdash A \wedge A}}{A \vdash A \wedge A}$$

# Using Structural Rules

$$\dfrac{\dfrac{A \vdash A}{A, B \vdash A}}{A \wedge B \vdash A}$$

$$\dfrac{\dfrac{A \vdash A \qquad A \vdash A}{A, A \vdash A \wedge A}}{A \vdash A \wedge A}$$

# Using Structural Rules

$$\dfrac{\dfrac{A \vdash A}{A, B \vdash A}}{A \wedge B \vdash A} \qquad \dfrac{A \vdash A \qquad A \vdash A}{A, A \vdash A \wedge A}}{A \vdash A \wedge A}$$

The usage of the contraction and weakening rules is crucial!

# Associativity

$$\frac{}{A \wedge (B \wedge C) \vdash (A \wedge B) \wedge C}$$

# Associativity

$$\dfrac{\dfrac{}{A, B, C \vdash (A \wedge B) \wedge C}}{A \wedge (B \wedge C) \vdash (A \wedge B) \wedge C}$$

# Associativity

$$\dfrac{\dfrac{\dfrac{}{A, B \vdash A \wedge B} \qquad C \vdash C}{A, B, C \vdash (A \wedge B) \wedge C}}{A \wedge (B \wedge C) \vdash (A \wedge B) \wedge C}$$

# Associativity

$$\dfrac{\dfrac{\dfrac{A \vdash A \qquad B \vdash B}{A, B \vdash A \wedge B} \qquad C \vdash C}{A, B, C \vdash (A \wedge B) \wedge C}}{A \wedge (B \wedge C) \vdash (A \wedge B) \wedge C}$$

# Alternative Rules for $\wedge$

We can replace $\wedge L$ with the following two rules:

$$\frac{\Gamma, A, B \vdash C}{\Gamma, A \wedge B \vdash C}\ \wedge\mathsf{L} \qquad \longrightarrow \qquad \frac{\Gamma, A \vdash C}{\Gamma, A \wedge B \vdash C}\ \wedge\mathsf{L}_1 \qquad \frac{\Gamma, B \vdash C}{\Gamma, A \wedge B \vdash C}\ \wedge\mathsf{L}_2$$

These rules **internalize weakening**.

# Alternative Rules for $\wedge$

We can replace $\wedge R$ with the following rule:

$$\frac{\Gamma_1 \vdash A \qquad \Gamma_2 \vdash B}{\Gamma_1, \Gamma_2 \vdash A \wedge B} \longrightarrow \frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B}$$

# Outline

# From Intuitionistic to Linear Logic

- Main idea: treat $\otimes$ the same as $\wedge$, minus contraction and weakening rules.
- The context $\Gamma$ then externalizes $\otimes$ on the meta-level:

$$\Gamma \vdash A \qquad \longrightarrow \qquad \bigotimes \Gamma \implies A$$

- We will recover contraction and weakening later with the ! **modality**.

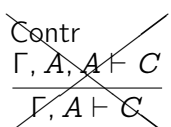
# Proof Theory of $\otimes$

Sequent calculus: $\Gamma \vdash A$

$$\frac{}{A \vdash A}$$

$$\otimes\mathrm{R}\ \frac{\Gamma_1 \vdash A \qquad \Gamma_2 \vdash B}{\Gamma_1, \Gamma_2 \vdash A \otimes B}$$

$$\otimes\mathrm{L}\ \frac{\Gamma, A, B \vdash C}{\Gamma, A \otimes B \vdash C}$$


$$\mathrm{eXchg}\ \frac{\Gamma, A, B, \Gamma' \vdash C}{\Gamma, B, A, \Gamma' \vdash C}$$

# Proof Theory of $\otimes$

Sequent calculus: $\Gamma \vdash A$

Left and right rules for $\otimes$

$$\frac{}{A \vdash A}$$

$$\otimes\mathrm{R}\ \frac{\Gamma_1 \vdash A \qquad \Gamma_2 \vdash B}{\Gamma_1, \Gamma_2 \vdash A \otimes B}$$

$$\otimes\mathrm{L}\ \frac{\Gamma, A, B \vdash C}{\Gamma, A \otimes B \vdash C}$$

$$\text{Weakn}\ \frac{\Gamma \vdash C}{\Gamma, A \vdash C}$$

$$\text{Contr}\ \frac{\Gamma, A, A \vdash C}{\Gamma, A \vdash C}$$

$$\text{eXchg}\ \frac{\Gamma, A, B, \Gamma' \vdash C}{\Gamma, B, A, \Gamma' \vdash C}$$

# Proof Theory of $\otimes$

Sequent calculus: $\Gamma \vdash A$

$$\frac{}{A \vdash A} \qquad \otimes\mathrm{R}\ \frac{\Gamma_1 \vdash A \qquad \Gamma_2 \vdash B}{\Gamma_1, \Gamma_2 \vdash A \otimes B} \qquad \otimes\mathrm{L}\ \frac{\Gamma, A, B \vdash C}{\Gamma, A \otimes B \vdash C}$$

$$\text{Weakn}\ \frac{\Gamma \vdash C}{\Gamma, A \vdash C} \qquad \text{Contr}\ \frac{\Gamma, A, A \vdash C}{\Gamma, A \vdash C} \qquad \text{eXchg}\ \frac{\Gamma, A, B, \Gamma' \vdash C}{\Gamma, B, A, \Gamma' \vdash C}$$

No weakening or contraction

# Adding the Unit

A proof theory with just $\otimes$ is not fun.
We need multiplicative unit (1) to signify **absence of resources**:

$$\frac{}{\emptyset \vdash 1} \qquad \frac{\Gamma \vdash C}{\Gamma, 1 \vdash C}$$

# Adding Implication

A proof theory with just $\otimes, 1$ is not fun.
We need linear implication ($\multimap$, *lollipop*) to form linear hypotheticals:

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B} \qquad \frac{}{\Gamma_1, \Gamma_2, A \multimap B \vdash C}$$

# Adding Implication

A proof theory with just $\otimes, 1$ is not fun.
We need linear implication ($\multimap$, *lollipop*) to form linear hypotheticals:

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B} \qquad \frac{\Gamma_1 \vdash A \qquad \Gamma_2, B \vdash C}{\Gamma_1, \Gamma_2, A \multimap B \vdash C}$$

Exercise: try to derive $(A \otimes B) \multimap C \dashv\vdash A \multimap (B \multimap C)$

# Cut rule

$$\frac{\Gamma \vdash A \qquad \Gamma', A \vdash B}{\Gamma, \Gamma' \vdash B}\ \text{Cut}$$

What is so special about this rule?
Each cut rule introduces a piece of "complexity" as a new formula $A$

# Cut rule

$$\frac{\Gamma \vdash A \qquad \Gamma', A \vdash B}{\Gamma, \Gamma' \vdash B}\ \text{Cut}$$

What is so special about this rule?
Each cut rule introduces a piece of "complexity" as a new formula $A$

## Theorem

*Every proof of* $\Gamma \vdash A$ *that uses cut can be converted into a proof that does not use cut*

# MILL

$$A, B \quad ::= \quad 1 \mid A \otimes B \mid A \multimap B$$

$$\frac{}{A \vdash A} \qquad \frac{}{\emptyset \vdash 1} \qquad \frac{\Gamma_1 \vdash A \qquad \Gamma_2 \vdash B}{\Gamma_1, \Gamma_2 \vdash A \otimes B}\ \otimes\mathsf{R} \qquad \frac{\Gamma, A, B \vdash C}{\Gamma, A \otimes B \vdash C}\ \otimes\mathsf{L}$$

$$\frac{\Gamma \vdash A \qquad \Gamma', A \vdash B}{\Gamma, \Gamma' \vdash B}\ \mathsf{Cut} \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B}\ \multimap\mathsf{R} \qquad \frac{\Gamma_1 \vdash A \qquad \Gamma_2, B \vdash C}{\Gamma_1, \Gamma_2, A \multimap B \vdash C}\ \multimap\mathsf{L}$$

# Adding &

Recall the different versions of left/right rules for $\wedge$

$$\wedge\mathrm{R}\quad \frac{\Gamma_1 \vdash A \qquad \Gamma_2 \vdash B}{\Gamma_1, \Gamma_2 \vdash A \wedge B}$$

$$\wedge\mathrm{R'}\quad \frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B}$$

$$\wedge\mathrm{L}\quad \frac{\Gamma, A, B \vdash C}{\Gamma, A \wedge B \vdash C}$$

$$\wedge\mathrm{L}_1\quad \frac{\Gamma, A_1 \vdash C}{\Gamma, A_1 \wedge A_2 \vdash C}$$

$$\wedge\mathrm{L}_2\quad \frac{\Gamma, A_2 \vdash C}{\Gamma, A_1 \wedge A_2 \vdash C}$$

# Adding &

Recall the different versions of left/right rules for $\wedge$

$$\otimes\text{R}\quad \frac{\Gamma_1 \vdash A \qquad \Gamma_2 \vdash B}{\Gamma_1, \Gamma_2 \vdash A \otimes B}$$

$$\wedge\text{R'}\quad \frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B}$$

$$\otimes\text{L}\quad \frac{\Gamma, A, B \vdash C}{\Gamma, A \otimes B \vdash C}$$

$$\wedge\text{L}_1\quad \frac{\Gamma, A_1 \vdash C}{\Gamma, A_1 \wedge A_2 \vdash C} \qquad \wedge\text{L}_2\quad \frac{\Gamma, A_2 \vdash C}{\Gamma, A_1 \wedge A_2 \vdash C}$$

# Adding &

Recall the different versions of left/right rules for $\wedge$

$$\otimes\mathrm{R}\ \frac{\Gamma_1 \vdash A \qquad \Gamma_2 \vdash B}{\Gamma_1, \Gamma_2 \vdash A \otimes B}$$

$$\&\mathrm{R}\ \frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \,\&\, B}$$

$$\otimes\mathrm{L}\ \frac{\Gamma, A, B \vdash C}{\Gamma, A \otimes B \vdash C}$$

$$\&\mathrm{L}_1\ \frac{\Gamma, A_1 \vdash C}{\Gamma, A_1 \,\&\, A_2 \vdash C} \qquad \&\mathrm{L}_2\ \frac{\Gamma, A_2 \vdash C}{\Gamma, A_1 \,\&\, A_2 \vdash C}$$

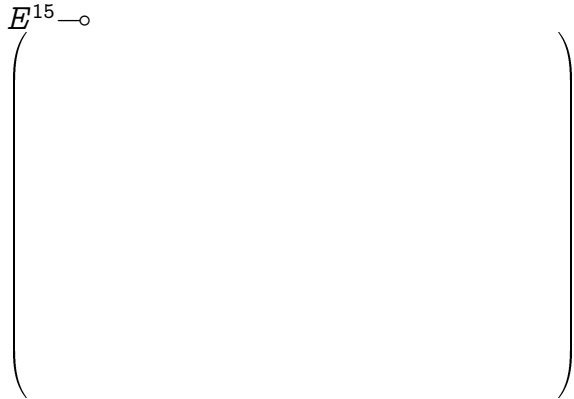
# Interplay of $\otimes$ and $\&$

- $A \,\&\, B \vdash A$, and $A \,\&\, B \vdash B$
- $A \,\&\, B \not\dashv\vdash A \otimes B$
- $(A \multimap B) \,\&\, (A \multimap C) \vdash A \multimap B \,\&\, C$
- $(A \,\&\, B) \multimap C \nvdash A \multimap (B \multimap C)$

# Outline

# Menu at the Linear Logic Cafe

**Summer Menu! €15 p.p.**
**Starter**: Greek Salad, or Soup
**Main**: Chicken Schnitzel, Tofu, or Salmon (for extra €2)
(all main dishes come with a side of fries)
**Desert**: Ice Cream, or Cheese Platter
**Drinks:** Leffe Tripel (for extra €3)



# Menu at the Linear Logic Cafe

**Summer Menu! €15 p.p.**
**Starter**: Greek Salad, or Soup
**Main**: Chicken Schnitzel, Tofu, or Salmon (for extra €2)
(all main dishes come with a side of fries)
**Desert**: Ice Cream, or Cheese Platter
**Drinks:** Leffe Tripel (for extra €3)

$$E^{15} \multimap \Big( (Sal \,\&\, Soup) \otimes \Big)$$

**Note:** We use & to indicate that the kitchen can provide any option you choose.

# Menu at the Linear Logic Cafe

**Summer Menu! €15 p.p.**
**Starter**: Greek Salad, or Soup
**Main**: Chicken Schnitzel, Tofu, or Salmon (for extra €2)
(all main dishes come with a side of fries)
**Desert**: Ice Cream, or Cheese Platter
**Drinks:** Leffe Tripel (for extra €3)

$$E^{15} \multimap \left( \begin{array}{l} (Sal \,\&\, Soup) \otimes \\ \Big( (Schn \,\&\, Tofu \,\&\, (E \otimes E \multimap Fish)) \end{array} \right)$$

**Note:** We use & to indicate that the kitchen can provide any option you choose.

# Menu at the Linear Logic Cafe

**Summer Menu! €15 p.p.**
**Starter**: Greek Salad, or Soup
**Main**: Chicken Schnitzel, Tofu, or Salmon (for extra €2)
(all main dishes come with a side of fries)
**Desert**: Ice Cream, or Cheese Platter
**Drinks:** Leffe Tripel (for extra €3)

$$E^{15} \multimap \left( \begin{array}{l} (Sal \,\&\, Soup) \otimes \\ \Big( (Schn \,\&\, Tofu \,\&\, (E \otimes E \multimap Fish)) \\ \qquad \otimes \; Fries \Big) \otimes \end{array} \right)$$

**Note:** We use & to indicate that the kitchen can provide any option you choose.

# Menu at the Linear Logic Cafe

**Summer Menu! €15 p.p.**
**Starter**: Greek Salad, or Soup
**Main**: Chicken Schnitzel, Tofu, or Salmon (for extra €2)
(all main dishes come with a side of fries)
**Desert**: Ice Cream, or Cheese Platter
**Drinks:** Leffe Tripel (for extra €3)

$$E^{15} \multimap \left( \begin{array}{l} (Sal \,\&\, Soup) \otimes \\ \Big( (Schn \,\&\, Tofu \,\&\, (E \otimes E \multimap Fish)) \\ \qquad \otimes \; Fries \Big) \otimes \\ (IceCr \,\&\, Cheese) \otimes \\ \\ \end{array} \right)$$

**Note:** We use & to indicate that the kitchen can provide any option you choose.

# Menu at the Linear Logic Cafe

**Summer Menu! €15 p.p.**
**Starter**: Greek Salad, or Soup
**Main**: Chicken Schnitzel, Tofu, or Salmon (for extra €2)
(all main dishes come with a side of fries)
**Desert**: Ice Cream, or Cheese Platter
**Drinks:** Leffe Tripel (for extra €3)

$$E^{15} \multimap \left( \begin{array}{l} (Sal \,\&\, Soup) \otimes \\ \Big( (Schn \,\&\, Tofu \,\&\, (E \otimes E \multimap Fish)) \\ \qquad \otimes \; Fries \Big) \otimes \\ (IceCr \,\&\, Cheese) \otimes \\ \\ (E^3 \multimap Beer) \end{array} \right)$$

**Note:** We use & to indicate that the kitchen can provide any option you choose.

# Grandma's Linear Logic Pizza recipe

**LL Pizza Ingredients:**

- Yeast and flour,
- Salt and sugar
- Tomatoes
- Sausage or paprika

$$\left( \quad \right) \multimap Pizza$$

# Grandma's Linear Logic Pizza recipe

**LL Pizza Ingredients:**

- Yeast and flour,
- Salt and sugar
- Tomatoes
- Sausage or paprika

$$\left(\begin{array}{l} Yeast \otimes Flour \otimes \\ \\ \\ \\ \\ \end{array}\right) \multimap Pizza$$

# Grandma's Linear Logic Pizza recipe

**LL Pizza Ingredients:**

- Yeast and flour,
- Salt and sugar
- Tomatoes
- Sausage or paprika

$$\left(\begin{array}{l} Yeast \otimes Flour \otimes \\ \\ Salt \otimes Sugar \otimes \\ \\ \\ \\ \\ \end{array}\right) \multimap Pizza$$

# Grandma's Linear Logic Pizza recipe

**LL Pizza Ingredients:**

- Yeast and flour,
- Salt and sugar
- Tomatoes
- Sausage or paprika

$$\left(\begin{array}{l} Yeast \otimes Flour \otimes \\ \\ Salt \otimes Sugar \otimes \\ \\ Tomatoes \otimes \\ \\ \end{array}\right) \multimap Pizza$$

# Grandma's Linear Logic Pizza recipe

**LL Pizza Ingredients:**

- Yeast and flour,
- Salt and sugar
- Tomatoes
- Sausage or paprika

$$\left(\begin{array}{l} Yeast \otimes Flour \otimes \\ \\ Salt \otimes Sugar \otimes \\ \\ Tomatoes \otimes \\ \\ (Sausage \text{ or } Paprika) \end{array}\right) \multimap Pizza$$

# Grandma's Linear Logic Pizza recipe

**LL Pizza Ingredients:**

- Yeast and flour,
- Salt and sugar
- Tomatoes
- Sausage or paprika

$$\left(\begin{array}{l} Yeast \otimes Flour \otimes \\ \\ Salt \otimes Sugar \otimes \\ \\ Tomatoes \otimes \\ \\ (Sausage \oplus Paprika) \end{array}\right) \multimap Pizza$$

**Note:** We use $\oplus$ to indicate that the grandma chooses (and we accept her choice).

# Adding $\oplus$

$$\oplus\mathsf{R}_1\ \frac{\Gamma \vdash A_1}{\Gamma \vdash A_1 \oplus A_2} \qquad \oplus\mathsf{R}_2\ \frac{\Gamma \vdash A_2}{\Gamma \vdash A_1 \oplus A_2} \qquad \oplus\mathsf{L}\ \frac{\Gamma, A_1 \vdash C \qquad \Gamma, A_2 \vdash C}{\Gamma, A_1 \oplus A_2 \vdash C}$$

# $\oplus$ vs $\&$

The rules are the same as for $\vee$ in the intuitionistic sequent calculus.

# $\oplus$ vs &

The rules are the same as for $\vee$ in the intuitionistic sequent calculus.

However,

$$E \,\&\, (B \oplus C) \nvdash (E \,\&\, B) \oplus (E \,\&\, C)$$

- $E =$ one euro;
- $B =$ beer, $C =$ coffee;
- $(B \oplus C) =$ either beer or coffee, but do not know which;
- $E \,\&\, (B \oplus C) =$ I can either have an euro or a drink.

# $\oplus$ vs $\&$

$$tea \oplus coffee \vdash awake$$

$$tea \,\&\, coffee \vdash awake$$

# $\oplus$ vs $\&$

$tea \oplus coffee \vdash awake$

Given either tea or coffee (I don't care which one), I can drink it and get awake

$tea \,\&\, coffee \vdash awake$

# $\oplus$ vs $\&$

$tea \oplus coffee \vdash awake$

Given either tea or coffee (I don't care which one), I can drink it and get awake

$tea \,\&\, coffee \vdash awake$

Given a choice of tea and coffee (for example at a hotel breakfast), I can drink a hot beverage and get awake

# $\oplus$ vs $\&$

$tea \oplus coffee \vdash awake$

Given either tea or coffee (I don't care which one), I can drink it and get awake

$tea \,\&\, coffee \vdash awake$

Given a choice of tea and coffee (for example at a hotel breakfast), I can drink a hot beverage and get awake

$tea \,\&\, coffee \vdash tea \oplus coffee$

# $\oplus$ vs $\&$

$tea \oplus coffee \vdash awake$

Given either tea or coffee (I don't care which one), I can drink it and get awake

$tea \,\&\, coffee \vdash awake$

Given a choice of tea and coffee (for example at a hotel breakfast), I can drink a hot beverage and get awake

$tea \,\&\, coffee \vdash tea \oplus coffee$

$A \,\&\, B \vdash A \oplus B$ $\qquad$ $A \oplus B \nvdash A \,\&\, B$

# MAILL

$$A, B \quad ::= \quad 1 \mid A \otimes B \mid A \multimap B \mid A \,\&\, B \mid A \oplus B$$

$$\frac{}{A \vdash A} \qquad \frac{}{\emptyset \vdash 1} \qquad \otimes\mathsf{R}\ \frac{\Gamma_1 \vdash A \qquad \Gamma_2 \vdash B}{\Gamma_1, \Gamma_2 \vdash A \otimes B} \qquad \otimes\mathsf{L}\ \frac{\Gamma, A, B \vdash C}{\Gamma, A \otimes B \vdash C}$$

$$\mathsf{Cut}\ \frac{\Gamma \vdash A \qquad \Gamma', A \vdash B}{\Gamma, \Gamma' \vdash B} \qquad \multimap\mathsf{R}\ \frac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B} \qquad \multimap\mathsf{L}\ \frac{\Gamma_1 \vdash A \qquad \Gamma_2, B \vdash C}{\Gamma_1, \Gamma_2, A \multimap B \vdash C} \qquad \&\mathsf{R}\ \frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \,\&\, B}$$

$$\&\mathsf{L}_i\ \frac{\Gamma, A_i \vdash C}{\Gamma, A_1 \,\&\, A_2 \vdash C} \qquad \oplus\mathsf{R}_i\ \frac{\Gamma \vdash A_i}{\Gamma \vdash A_1 \oplus A_2} \qquad \oplus\mathsf{L}\ \frac{\Gamma, A_1 \vdash C \qquad \Gamma, A_2 \vdash C}{\Gamma, A_1 \oplus A_2 \vdash C}$$

# Related Logics

- Classical Linear Logic

# Related Logics

- Classical Linear Logic
- Relevance logics
  - Contraction but no weakening
  - Intuition: all hypothesis are relevant and must be used

# Related Logics

- Classical Linear Logic
- Relevance logics
  - Contraction but no weakening
  - Intuition: all hypothesis are relevant and must be used
- Logic of Bunched Implications
  - Better known as the kernel of Separation Logic
  - Popular in deductive program verification

# Taking Stock

Up to here:

- Correctness for communicating programs
- Sessions as protocol specifications
- A formal syntax for session types
- Example: A two-buyer protocol
- Protocol compatibility in terms of duality for session types
- **Intuitionistic Linear Logic (MAILL)**

**Tomorrow**:

- Interpreting ILL as session types for message-passing processes