Verificación de Programas con F*

Clase 2 – 19/03/2024

¿Cuál es el tipo de printf?

```
printf “%s\n” : string -> unit
printf “%d\n” : int -> unit
printf “%d + %d = %f\n” : int -> int -> float -> unit
```

Entonces `printf : string -> …?`

Tipado simple

$$t ::= c \mid t \to t$$

$$\textsc{T-Abs-St}\quad \frac{\Gamma, x : t \vdash e \; : \; s}{\Gamma \vdash (\lambda(x : t).e) \; : \; t \to s}$$

$$\textsc{T-App-St}\quad \frac{\Gamma \vdash f \; : \; t \to s \qquad \Gamma \vdash e \; : \; t}{\Gamma \vdash f e \; : \; s}$$

- Tipado en STLC
- Los tipos son *cerrados*: no dependen del contexto Γ
- Si `f : t -> s`, entonces `f e` <u>debe</u> tener tipo s

Sistema F: polimorfismo paramétrico

$$t ::= X \mid c \mid t \to t \mid \forall X.t$$

$$\text{T-Abs}\ \frac{\Gamma, x : t \vdash e \;:\; s}{\Gamma \vdash (\lambda(x : t).e) \;:\; t \to s}$$

$$\text{T-App}\ \frac{\Gamma \vdash f \;:\; t \to s \qquad \Gamma \vdash e \;:\; t}{\Gamma \vdash fe \;:\; s}$$

$$\text{T-TAbs}\ \frac{\Gamma, X \vdash e \;:\; s}{\Gamma \vdash (\Lambda X.e) \;:\; \forall X.s}$$

$$\text{T-TApp}\ \frac{\Gamma \vdash e \;:\; \forall X.t \qquad \Gamma \vdash A \;:\; \mathit{Type}}{\Gamma \vdash eA \;:\; t[A/X]}$$

- Algo de “dependencia”: los términos pueden depender de tipos
  - a.k.a. polimorfismo, y en este caso es *paramétrico*
- Los tipos ahora *tienen variables ligadas*
  - Deben estar bien formados
- Dos “tipos” de aplicación y abstracción
  - Sintaxis distinguida

Wadler 1989 – Theorems for Free!

Tipado dependiente

Sintaxis uniforme para expresiones y tipos

$$t ::= x \mid c \mid (x : t) \to t$$

$(t \to t'$ es azúcar para $(x : t) \to t'$ con $x \notin FV(t'))$

x puede aparecer en **s**

$$\textsc{T-Abs-Dep}\quad \frac{\Gamma, x : t \vdash e \;:\; s}{\Gamma \vdash (\lambda(x : t).e) \;:\; (x : t) \to s}$$

$$\textsc{T-App-Dep}\quad \frac{\Gamma \vdash f \;:\; (x : t) \to s \qquad \Gamma \vdash e \;:\; t}{\Gamma \vdash f e \;:\; s[e/x]}$$

Sustitución por el argumento real

- En tipado dependiente, los tipos pueden depender de valores
  - Ya vimos algunos ejemplos encubiertos:

```
val incr''': (x:nat) -> y:int{y = x+1}
let incr''' (x:nat) : y:int{y = x+1} = x+1
```

Polimorfismo en F*

- Como en otros lenguajes con tipos las funciones polimórficas son simplemente funciones que toman un tipo como argumento

```
val id : (a:Type) -> a -> a
let id a x = x
```

```
let _ = assert (id int 42 == 42)
let _ = assert (id string "hola" == "hola")
```

```
let _ = assert (id Type string == string)
let _ = assert (id (id Type string) "hola" == "hola")
```

Al margen: terminología

$$\Pi_{x:A} B$$
$$(x : A) \to B$$
$$\Pi(x : A) B(x)$$

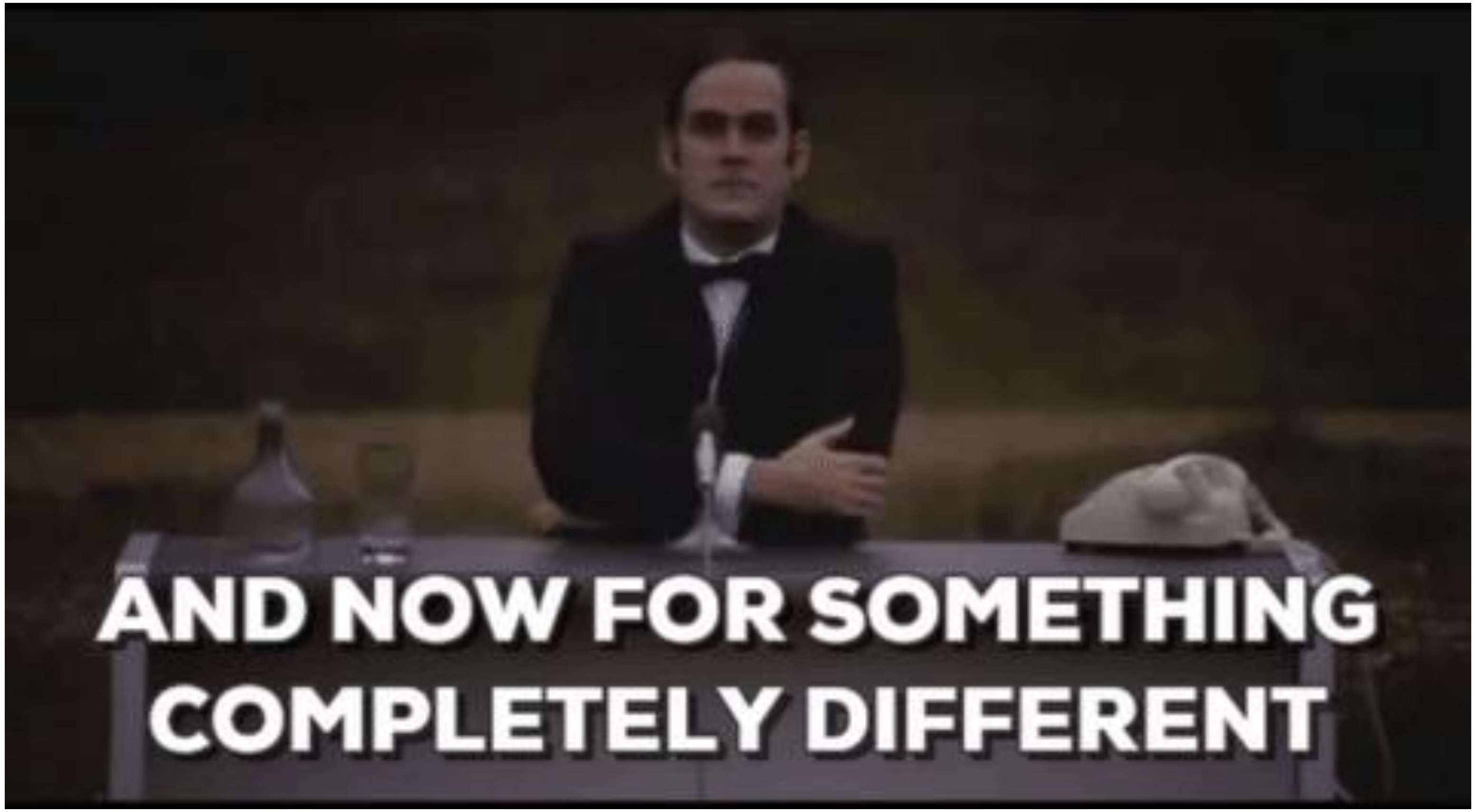
"Producto dependiente" (función dependiente, flecha dependiente, tipo Π)

$$\begin{array}{rcl}
\mathbf{Bool} \to A & \cong & A \times A \\
\mathbf{Nat} \to A & \cong & A \times A \times \cdots \times A \\
\Pi_{x:\mathbf{Nat}} A(x) & \cong & A(0) \times A(1) \times \cdots \times A(n) \times \cdots
\end{array}$$

"Suma dependiente" (par dependiente, tupla dependiente, tipo Σ)

$$\begin{array}{rcl}
\mathbf{Bool} \times A & \cong & A + A \\
\mathbf{Nat} \times A & \cong & A + A + \cdots + A \\
\Sigma_{x:\mathbf{Nat}} A(x) & \cong & A(0) + A(1) + \cdots + A(n) + \cdots
\end{array}$$

(demo)


AND NOW FOR SOMETHING
COMPLETELY DIFFERENT

Lógica formal

- Presentación "a la Gentzen"
  - Hipótesis "a descargar"
- Reglas de introducción y eliminación
- Una prueba es una derivación correcta de una proposición
- "Formal": las pruebas pueden chequearse mecánicamente

$$\Longrightarrow\textsc{-Intro}\quad \dfrac{\begin{array}{c}A\\ \vdots\\ B\end{array}}{A \Longrightarrow B} \qquad \Longrightarrow\textsc{-Elim}\quad \dfrac{A \Longrightarrow B \qquad A}{B}$$

$$\wedge\textsc{-Intro}\quad \dfrac{A \qquad B}{A \wedge B} \qquad \wedge\textsc{-Elim-L}\quad \dfrac{A \wedge B}{A} \qquad \wedge\textsc{-Elim-R}\quad \dfrac{A \wedge B}{B}$$

Lógica formal - Secuentes

- Secuentes: explicitan las hipótesis
- Prop de Eliminación de cortes:
  - Corte: introducción seguida de eliminación para un mismo conectivo
  - Toda prueba puede simplificarse a una prueba sin cortes
- Todo corte puede reducirse, eso es trivial
- ¿El proceso termina?
  - Sí, Gentzen demostró que si se toman los cortes más externos, siempre termina

$$\frac{}{\Gamma, A, \Gamma' \vdash A}\ \textsc{Ax} \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \implies B}\ {\implies}\text{-}\textsc{Intro}$$

$$\frac{\Gamma \vdash A \implies B \qquad \Gamma \vdash A}{\Gamma \vdash B}\ {\implies}\text{-}\textsc{Elim}$$

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\ \wedge\text{-}\textsc{Intro} \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}\ \wedge\text{-}\textsc{Elim-L} \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}\ \wedge\text{-}\textsc{Elim-R}$$

$$\dfrac{\dfrac{\dfrac{\begin{matrix}[\Gamma, A \vdash A] \\ \vdots\end{matrix}}{\Gamma, A \vdash^1 B}}{\Gamma \vdash A \implies B} \qquad \Gamma \vdash^2 A}{\Gamma \vdash B} \quad \rightsquigarrow \quad \dfrac{\begin{matrix}[\Gamma \vdash^2 A] \\ \vdots\end{matrix}}{\Gamma \vdash^1 B}$$

Girard 1987 – "Una lógica sin eliminación de cortes es como un auto sin motor"

$$\frac{}{\Gamma, A, \Gamma' \vdash A}\ \textsc{Ax} \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \implies B}\ {\implies}\textsc{-Intro}$$

$$\frac{\Gamma \vdash A \implies B \qquad \Gamma \vdash A}{\Gamma \vdash B}\ {\implies}\textsc{-Elim}$$

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\ \wedge\textsc{-Intro} \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}\ \wedge\textsc{-Elim-L} \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}\ \wedge\textsc{-Elim-R}$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\ \vee\textsc{-Intro-L} \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\ \vee\textsc{-Intro-R} \qquad \frac{\Gamma \vdash A \vee B \qquad \Gamma, A \vdash C \qquad \Gamma, B \vdash C}{\Gamma \vdash C}\ \vee\textsc{-Elim}$$

$$\frac{}{\Gamma \vdash \top}\ \top\textsc{-Intro} \qquad \frac{\Gamma \vdash \bot}{\Gamma \vdash A}\ \bot\textsc{-Elim}$$

(PD: Consistencia)

- ¿Cómo se demuestra que la lógica es *consistente*?
  - i.e. tiene al menos una prop P tal que **no** ocurre $\vdash P$
- 1. Se demuestra que la lógica tiene la propiedad de eliminación de cortes
  - Esta es la parte difícil, en general por mostrar que el procedimiento termina
- 2. No hay pruebas sin cortes de Falso
  - Una prueba sin cortes tiene que terminar con una introducción
  - Entonces esto es trivial, porque falso no tiene introducciones. $\nvdash \bot$

Correspondencia Curry-Howard

- a.k.a. “propositions as types”

$$\textsc{Ax}\ \frac{}{\Gamma, A, \Gamma' \vdash A} \qquad \Longrightarrow\text{-}\textsc{Intro}\ \frac{\Gamma, A \vdash B}{\Gamma \vdash A \implies B}$$

$$\Longrightarrow\text{-}\textsc{Elim}\ \frac{\Gamma \vdash A \implies B \qquad \Gamma \vdash A}{\Gamma \vdash B}$$

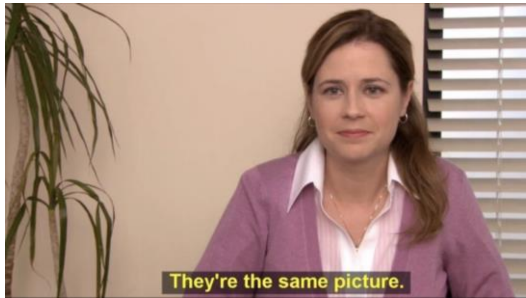
$$\wedge\text{-}\textsc{Intro}\ \frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \qquad \wedge\text{-}\textsc{Elim-L}\ \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \qquad \wedge\text{-}\textsc{Elim-R}\ \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}$$

$$\textsc{Ax}\ \frac{}{\Gamma, x : A, \Gamma' \vdash x : A} \qquad \to\text{-}\textsc{Intro}\ \frac{\Gamma, x : A \vdash e : B}{\Gamma \vdash (\lambda x.e) : A \to B}$$

$$\to\text{-}\textsc{Elim}\ \frac{\Gamma \vdash f : A \to B \qquad \Gamma \vdash e : A}{\Gamma \vdash fe : B}$$

$$\times\text{-}\textsc{Intro}\ \frac{\Gamma \vdash x : A \qquad \Gamma \vdash y : B}{\Gamma \vdash (x, y) : A \times B} \qquad \times\text{-}\textsc{Elim-L}\ \frac{\Gamma \vdash p : A \times B}{\Gamma \vdash \textbf{fst}\ p : A} \qquad \times\text{-}\textsc{Elim-R}\ \frac{\Gamma \vdash A \times B}{\Gamma \vdash \textbf{snd}\ p : B}$$

Relacionado: interpretación Brouwer-Heyting-Kolmogorov de lógica intuicionista



Curry-Howard

- Tomamos una proposición (tipo) P como cierta si existe un habitante de P
- Las pruebas **son programas**, reducen y tienen contenido computacional
  - Una prueba de A \/ B decide cuál caso vale
  - Cuando tengamos existenciales: son programas que computan un testigo

$$\dfrac{\dfrac{\dfrac{}{p : A \times B \vdash p : A \times B}}{p : A \times B \vdash \mathbf{snd}\ p : B} \qquad \dfrac{\dfrac{}{p : A \times B \vdash p : A \times B}}{p : A \times B \vdash \mathbf{fst}\ p : A}}{\dfrac{p : A \times B \vdash (\mathbf{snd}\ p, \mathbf{fst}\ p)\ :\ B \times A}{\vdash \lambda p.(\mathbf{snd}\ p, \mathbf{fst}\ p)\ :\ A \times B \to B \times A}}$$

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

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

Curry-Howard

- El término codifica la prueba
  - Si el tipado es dirigido por sintaxis, es inmediato construir un tipado para un término dado
  - Chequear una prueba es fácil = chequear el tipo de un término es fácil
  - Encontrar una prueba es difícil = encontrar un habitante de un tipo es difícil
- Es un principio general, hay una correspondencia para cada lógica
  - STLC ≈ Lógica proposicional
  - System F ≈ Lógica de segundo orden
  - Tipado dependiente (tipos dependen de valores) ≈ Lógica de predicados
  - Tipado dependiente + polimorfismo ≈ lógica de alto orden
  - Y más…

$$\dfrac{\dfrac{\dfrac{\overline{p : A \times B \vdash p : A \times B}}{p : A \times B \vdash \mathbf{snd}\ p : B} \qquad \dfrac{\overline{p : A \times B \vdash p : A \times B}}{p : A \times B \vdash \mathbf{fst}\ p : A}}{p : A \times B \vdash (\mathbf{snd}\ p, \mathbf{fst}\ p)\ :\ B \times A}}{\vdash \lambda p.(\mathbf{snd}\ p, \mathbf{fst}\ p)\ :\ A \times B \to B \times A}$$

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

$$\dfrac{\dfrac{\dfrac{\overline{A \wedge B \vdash A \wedge B}}{A \wedge B \vdash B} \qquad \dfrac{\overline{A \wedge B \vdash A \wedge B}}{A \wedge B \vdash A}}{A \wedge B \vdash B \wedge A}}{\vdash A \wedge B \implies B \wedge A}$$

Correspondencia Curry-Howard (2)

- La correspondencia se extiende a la simplificación de pruebas
  - Tipo ≈ proposición o predicado
  - Prueba ≈ programa/algoritmo
  - “cut-elimination” de la implicancia ≈ beta-reducción
  - “cut-elimination” de pares ≈ reducción de proyecciones (fst (x,y) ~> x)
  - Prueba en forma normal ≈ término en forma normal
  - La lógica tiene eliminación de cortes ≈ el lenguaje es débilmente normalizante (en general… hay casos borde)

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}[\Gamma, A \vdash A]\\ \vdots\end{array}}{\Gamma, A \vdash^1 B}}{\Gamma \vdash A \implies B} \qquad \Gamma \vdash^2 A}{\Gamma \vdash B} \quad \rightsquigarrow \quad \dfrac{\begin{array}{c}[\Gamma \vdash^2 A]\\ \vdots\end{array}}{\Gamma \vdash^1 B}$$

["Propositions as Types" by Philip Wadler - YouTube]

Intuicionismo / constructivismo

- Para capturar la lógica clásica, nos falta una regla importante

$$\frac{}{\Gamma \vdash A \vee \neg A}$$

- En lógica intuicionista, se rechaza este axioma
  - Implicaría un procedimiento de decisión para cada prop A (en lógicas suf. expresivas contradice a Gödel y Turing a la vez)
  - No tiene interpretación computacional
  - Rompe propiedades como $(\vdash A \vee B) \implies (\vdash A) \vee (\vdash B)$
- Hay más axiomas clásicos, muchos son equivalentes
  - Notoriamente la eliminación de doble negación: $\neg\neg A \implies A$
  - Otros: ver práctica

Prueba no constructivas

Veamos un ejemplo. *Existen irracionales a y b, tales que a^b es racional.*

Demostración (por el absurdo): consideremos $\sqrt{2}^{\sqrt{2}}$. Por el principio del tercero excluido, este número será racional o irracional. En el primer caso, basta tomar $a = b = \sqrt{2}$; en el segundo caso, basta tomar $a = \sqrt{2}^{\sqrt{2}}$ y $b = \sqrt{2}$, pues ${\sqrt{2}^{\sqrt{2}}}^{\sqrt{2}} = 2$.

La demostración es impecable, pero no sabemos si $\sqrt{2}^{\sqrt{2}}$ es racional o no.

De la charla “Continuaciones: La Venganza del Goto” de Guido Macchi, JCC 2007

(demo)

Pureza / totalidad

- En un lenguaje de programación, ¿por qué no podemos demostrar cualquier P mediante recursion?

```
let rec prueba () : p = prueba ()
let prueba () : p = let x = 1/0 in …
let prueba () : p = raise "☺"
```

- En la presencia de **efectos** (no terminación, parcialidad, excepciones, etc), la correspondencia se rompe. En general, sólo vale para lenguajes puros
- En F* tenemos definiciones puras y *efectuosas*
  - El sistema de efectos se encarga de separar las efectuosas del fragmento puro
  - Implica chequear terminación, completitud de pattern match, etc…

Tareas

- Completar Clase2.fst
- Leer capítulos 3 y 6

- Otras fuentes:
  - "Propositions as Types" by Philip Wadler - YouTube
  - Cayenne – a language with dependent types (Augustsson 1998)
  - El tipo de printf: printf* (fstarlang.github.io)
  - Continuations and Logic.pdf (cmu.edu)
  - Guido Macchi – Continuaciones la Venganza del Goto (JCC 2007)