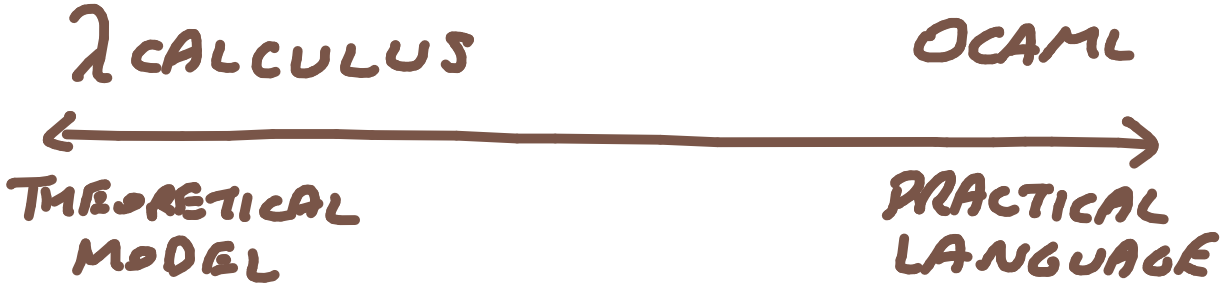
TYPE SYSTEMS

I MENTIONED THAT THE LAMBDA CALCULUS WAS THE FOUNDATION OF THE OCAML PROGRAMMING LANGUAGE (BY WAY OF LISP, AMONG OTHERS)

WORKING ON THE HOMEWORK SHOULD HAVE REINFORCED THAT RELATIONSHIP, SINCE THE CODE YOU ENDED UP WRITING WAS VERY OCAML-LIKE.

WHILE RELATED, THE λ CALCULUS AND OCAML LIVE AT TWO ENDS OF A SPECTRUM



WE ARE GOING TO TALK ABOUT TYPE SYSTEMS TODAY, OF THE KIND FOUND IN OCAML, BY ADDING A TYPE SYSTEM TO THE λ CALCULUS.

TO DO SO, WE FIRST NEED TO SHIFT THE λ CALCULUS A BIT MORE TO THE RIGHT.

WE ADDED THINGS LIKE NATURAL NUMBERS, PAIRS, AND THE LIKES TO THE λ CALCULUS BY WAY OF ENCODINGS. THAT'S GREAT FOR A THEORETICAL MODEL WHERE WE WANT TO SHOW THAT THOSE THINGS ARE JUST DERIVED NOTIONS, BUT NOT SO MUCH FOR MODELING A PRACTICAL LANGUAGE.

SO WE ADD THE NOTION OF A "VALUE" TO THE λ CALCULUS, WITH PRIMITIVE VALUES LIKE INTEGERS AND BOOLEANS AND ASSOCIATED OPERATIONS, TO FORM AN "APPLIED" λ CALCULUS.

A TERM* OF THE APPLIED λ CALCULUS IS ONE OF:

(* ALSO CALLED AN EXPRESSION)

- AN IDENTIFIER $x, y, z, \ldots$
- A VALUE V
- AN APPLICATION $M\ N$ (WHERE M, N ARE TERMS)
- AN EXPRESSION $M + N$ OR $M * N$ (WHERE M, N ARE TERMS)
- A CONDITIONAL IF M THEN N_1 ELSE N_2 (WHERE M, N_1, N_2 ARE TERMS)

MEANWHILE, A VALUE IS ONE OF:

- AN INTEGER $i \in \mathbb{Z}$
- A BOOLEAN TRUE OR FALSE
- A FUNCTION $(x \to M)$
  (WHERE x IS AN IDENTIFIER AND M A TERM)

THIS CAN BE WRITTEN MORE SUCCINCTLY USING A BNF GRAMMAR

$$
\begin{aligned}
M, N ::= {} & x && x \in \text{IDENT} \\
& V \\
& M\ N \\
& M + N \\
& M * N \\
& \text{IF } M \text{ THEN } N_1 \text{ ELSE } N_2
\end{aligned}
$$

$$
\begin{aligned}
V ::= {} & i && i \in \mathbb{Z} \\
& \text{TRUE} \\
& \text{FALSE} \\
& (x \to M) && x \in \text{IDENT}
\end{aligned}
$$

SIMPLIFICATION IN THE APPLIED λ CALCULUS, CALLED EVALUATION, IS DONE VIA THE USUAL SIMPLIFICATION RULE, PLUS ADDITIONAL RULES FOR THE NEW OPERATIONS +, *, AND IF

$$(x \to M)\ N \Rightarrow M\{N/x\}$$

$$i_1 + i_2 \Rightarrow i_3 \quad (i_3 = i_1 + i_2)$$
$$i_1 * i_2 \Rightarrow i_3 \quad (i_3 = i_1 \cdot i_2)$$

IF TRUE THEN N_1 ELSE $N_2 \Rightarrow N_1$
IF FALSE THEN N_1 ELSE $N_2 \Rightarrow N_2$

IN THE APPLIED λ CALCULUS, THE GOAL IS TO TAKE AN EXPRESSION AND EVALUATE IT (SIMPLIFY IT) UNTIL YOU GET A VALUE. THERE MAY BE TWO REASONS YOU CANNOT — YOU ALWAYS CAN EVALUATE, OR YOU GET STUCK ON A TERM YOU CAN NO LONGER EVALUATE BUT IS ALSO NOT A VALUE, LIKE

IF a THEN 1 ELSE 2

(SINCE WE HAVE NO RULE FOR HOW TO SIMPLIFY AN IF WHEN THE CONDITION IS THE IDENTIFIER a, WE ARE STUCK.)

TYPE SYSTEMS AIM AT RULING OUT EXPRESSIONS THAT WOULD GET STUCK DURING EVALUATION

A TYPE IS A SET OF VALUES

E.G., INT IS THE SET OF ALL INTEGERS.

A VALUE V HAS TYPE T IF, BASICALLY, $V \in T$.

A TYPE SYSTEM IS A MECHANISM (USUALLY BASED ON RULES) THAT ASSIGNS A TYPE TO EVERY EXPRESSION.

WE CARE ABOUT TYPE SYSTEMS THAT ARE SOUND: IF THE TYPE SYSTEM ASSIGNS TYPE T TO AN EXPRESSION, THEN IF EVALUATION OF THE EXPRESSION TERMINATES, IT YIELDS A VALUE OF TYPE T. IN PARTICULAR, EVALUATION DOES NOT GET STUCK.

(IF THE TYPE SYSTEM CAN ASSIGN A TYPE TO THE EXPRESSION, WE SAY THE EXPRESSION TYPE CHECKS.

IF NOT, THEN IT HAS A TYPE ERROR.)

OCAML USES A SPECIFIC TYPE SYSTEM CALLED THE HINDLEY-MILNER TYPE SYSTEM. (WELL, A VARIATION THEREOF.)

WE ARE GOING TO DEVELOP A SIMPLER TYPE SYSTEM FOR OUR APPLIED λ CALCULUS — THE SIMPLY-TYPED λ CALCULUS.

WE MERELY NEED TO MODIFY OUR SYNTAX FOR FUNCTIONS TO ADD A TYPE ANNOTATION, AND ADD SOME SYNTAX FOR TYPES:

$$V ::= \ldots \mid (x : T \to M) \qquad x \in \text{IDENT}$$

$$T, U ::= \text{INT} \mid \text{BOOL} \mid T \to U \qquad \} \text{ TYPES}$$

THE TYPE OF FUNCTIONS TAKING VALUES OF TYPE T AND YIELDING VALUES OF TYPE U IS $T \to U$.

HERE IS A FUNCTION OF TYPE INT→INT:

$$(x : \text{INT} \longrightarrow x + 1)$$

THE TYPE SYSTEM IS DEFINED VIA A FUNCTION TYPEOF THAT TAKES:

- AN EXPRESSION
- AN ENVIRONMENT Γ (A MAP FROM IDENTIFIERS TO TYPES, GIVING THE TYPES OF THE AVAILABLE NAMES)

AND EITHER RETURNS THE TYPE ASSIGNED TO THE EXPRESSION OR FAILS IF NO SUCH TYPE CAN BE FOUND.

TYPEOF IS DEFINED RECURSIVELY OVER THE FORM OF THE SYNTAX OF EXPRESSIONS:

$$\text{TYPEOF}(x, \Gamma) = \Gamma(x)$$

$$\text{TYPEOF}(M\,N, \Gamma) = \begin{cases} U \text{ IF } \text{TYPEOF}(M,\Gamma) = T \to U \\ \quad \text{AND } \text{TYPEOF}(N,\Gamma) = T \\ \text{FAIL} \quad \text{OTHERWISE} \end{cases}$$

$$\text{TYPEOF}(M+N,\Gamma) \\ \text{TYPEOF}(M*N,\Gamma) = \begin{cases} \text{INT IF } \text{TYPEOF}(M,\Gamma) \\ \quad = \text{TYPEOF}(N,\Gamma) = \text{INT} \\ \text{FAIL} \quad \text{OTHERWISE} \end{cases}$$

$$\text{TYPEOF}(\underline{\text{IF}}\ M\ \underline{\text{THEN}}\ N_1\ \underline{\text{ELSE}}\ N_2, \Gamma) = \begin{cases} T & \text{IF TYPEOF}(M,\Gamma) = \text{BOOL} \\ & \text{AND TYPEOF}(M,\Gamma) = \text{TYPEOF}(N_1,\Gamma) \\ & = T \\ \underline{\text{FAIL}} & \text{OTHERWISE} \end{cases}$$

$$\text{TYPEOF}(i, \Gamma) = \text{INT}$$

$$\text{TYPEOF}(\underline{\text{TRUE}}, \Gamma) = \text{BOOL}$$
$$\text{TYPEOF}(\underline{\text{FALSE}}, \Gamma) = \text{BOOL}$$

$$\text{TYPEOF}((\lambda x{:}T \to M), \Gamma) = \begin{cases} T \to U & \text{if TYPEOF}(M, \Gamma[x \mapsto T]) \\ \underline{\text{FAIL}} & \text{OTHERWISE} \end{cases}$$

(WHERE $\Gamma[x \mapsto T]$ IS THE MAP THAT ACTS LIKE Γ EXCEPT THAT IT MAPS x TO T.)

THEOREM: IF $\text{TYPEOF}(M, \emptyset) = T$, THEN EVALUATING M TERMINATES AND YIELDS A VALUE OF TYPE T.

THUS THE SIMPLY-TYPED λ CALCULUS IS SOUND (YAY!) BUT IS NOT TURING COMPLETE (BOO!) — WE CANNOT WRITE INFINITE LOOPS, SO WE CANNOT SIMULATE ARBITRARY TURING MACHINES

NOTE THAT

$$Y = (f \to (x \to f(x\,x))(x \to f(x\,x)))$$

DOES NOT TYPE CHECK IN THE SIMPLY-TYPED λ CALCULUS (WHAT TYPE WOULD YOU ASSIGN x IN $(x \to f(x\,x))$??)

TO RESTORE TURING COMPLETENESS YOU COULD SIMPLY ADD A FIXED POINT OPERATOR TO THE CALCULUS:

$$M ::= \ldots \mid \underline{\text{Fix}}\ V$$

WHERE $\underline{\text{Fix}}\ V \Rightarrow V(\underline{\text{Fix}}\ V)$

AND $\text{TYPEOF}(\text{Fix}\ V, \Gamma) = \begin{cases} T \text{ IF } \text{TYPEOF}(V, \Gamma) = T \to T \\ \underline{\text{FAIL}} \text{ OTHERWISE} \end{cases}$

FOR INSTANCE, IF YOU ADD PRIMITIVE OPERATIONS

$M ::= \ldots$
$\quad$ ZERO? M
$\quad M - N$

WITH EVALUATION RULES

ZERO? $0 \Longrightarrow$ TRUE
ZERO? $i \longrightarrow$ FALSE $\quad (i \neq 0)$
$i_1 - i_2 \Longrightarrow i_3 \quad (i_3 = i_1 - i_2)$

AND TYPEOF (ZERO? M, Γ)
$= \begin{cases} \text{BOOL IF TYPEOF}(M,\Gamma) = \text{INT} \\ \underline{\text{FAIL}} \text{ OTHERWISE} \end{cases}$

TYPEOF ($M - N$, Γ)
$= \begin{cases} \text{INT IF TYPEOF}(M,\Gamma) = \text{TYPEOF}(N,\Gamma) = \text{INT} \\ \underline{\text{FAIL}} \text{ OTHERWISE} \end{cases}$

THEN YOU CAN DEFINE

FACT = $\underline{\text{FIX}}$ (f: INT→INT →
$\quad$ (n: INT → $\underline{\text{IF}}$ (ZERO? n)
$\quad\quad$ $\underline{\text{THEN}}$ 1
$\quad\quad$ $\underline{\text{ELSE}}$ n * (f (n - 1)))))

AND CHECK:

TYPEOF (FACT, $\emptyset$) = INT → INT