1 What is curry-howard correspondence?

.
In programming language theory and proof theory, the Curry-Howard correspondence (also known as the Curry-Howard isomorphism or equivalence, or the proofs-as-programs and propositions- or formulae-as-types interpretation) is the direct relationship between computer programs and mathematical proofs.

1.1 type constructions in FP

But before getting into that let's see type constructions in functional programming *Here I'll use SML as an example*

- Tuple type `(int,string)`
  Creating : `val x : int*string = (4 , ‘‘Hello’’)`
  Using : `val y : int = #1 x`

- Function type `int -> string`
  Creating : `fun f (x:int) :string = ‘‘the value of x is’’ ^ toString(x)`
  Using : `val y: string = f 45`

- Disjunction type `datatype X = Left of int | Right of string`
  Creating : `val x:X = Left 4`
  Creating : `val y:X = Right ‘‘Hello’’`
  Using :
  ```
  fun f (x:X) :bool = case x of
                      Left y = ( y > 0 )
                      Right _ = false
  ```
  the above given is a function that takes an argument of type X and returns a value of type bool

- Unit type `type t = unit`
  Creating : `val x:unit = ()`

1.2 From types to propositions

But how does the types defined just translate themselves into propositions? consider the sml code `val x:t = .....` now what does this mean? If this code manages to compile without error then there the type t exists in the system. Let's denote this proposition by $CH\,(t)$ which means that code has a value of type t or the type t exists in the system.
But that was just for a type t what about all the other things that we have discussed above?

Type	Proposition	Short Notation
t	CH(t)	t
(a,b)	CH(a) and CH(b)	A $\wedge$ B ; A $\times$ B
left a | right b	CH(a) or CH(b)	A $\vee$ B ; A + B
a $\rightarrow$ b	CH(a) implies CH(b)	A $\Longrightarrow$ B
unit	true	1

- type parameter `'a` can be considered as $\forall a$
- consider the function `fun f (x: 'a) : 'a * 'a = (x,x)` this is logically equivalent to $\forall a \quad a \Longrightarrow a \wedge a$
- The elementary proof task is represented by a sequent. Notation $A, B, C \vdash G$ the premises are A,B and C and the goal is G.
- proofs are achieved through axioms and derivation rules. Axioms : this sequent is true. Derivation: if such sequents are true then other such sequents are true.
- consider the SML expression `val x = 3; val a = Int.toString(x:int) ^ ''abc'';` an expression of type String This is equivalent to the sequent $Int \vdash String$. This sequent uses the fact that it uses the already computed expression x which is an integer. and hence that becomes the premise and what we are computing here is the value of string which becomes the goal.
- Now consider the function `fun f (x:int) :string = Int.toString(x) ^ ''abc''` . this takes an argument of integer type and returns a value of string type.This doesn't assume that x is defined outside. this uses x as an argument . and it is represented by the sequent $\phi \vdash Int \Rightarrow String$. The empty set on the premises implies that this expression doesn't use any variables that are previously computed. and the goal is the function type.
- sequents only describe the types and doesn't say anything about the value that is being computed.

Now lets see how the type constructions stated above translates into axioms and derivation rules.

- the tuple type. $A \times B$
  Creation : $A, B \vdash A \times B$. that is if we have types A and B then we have type $A \times B$.
  Usage: $A \times B \vdash A$ or $A \times B \vdash B$.
- Function type . $A \Rightarrow B$.
  Creation : If $A \vdash B$ then we have $\phi \vdash A \Rightarrow B$.
  Usage : $A \Rightarrow B, A \vdash B$.

- Disjunction type $A + B$:
  Creation : $A \vdash A + B$ and also $B \vdash A + B$
  Usage : $A + B, A \Rightarrow C, B \Rightarrow C \vdash C$. this corresponds to the function (that uses case statements) that we've written in the SML example for disjunction.

- Unit type 1:
  Create : $\phi \vdash 1$