Dependent Type Programming Paradigm

Constant problem

- Writing a correct computer program is hard and proving that the program is correct is even harder.
- Dependent types allow us to write programs and know they are correct before running them.
- Dependent types: specify types that can check the value of the variables at compile time.

Example:

Declare a vector that contains the value 1,2,3:

```
val l1= 1:#: 2 :#: 3:#: Vnil
```

Vector length is 3

```
val l1= 1:#: 2 :#: 3:#: Vnil
val l2= 1:#: 2 :#: 3:#: Vnil
val l3= l1 vAdd l2
```

Output:

```
l3= 2:#:4:#:6:#:VNil
```

The example above works fine because the type system knows both vectors have length 3

```
val l1= 1:#: 2 :#: 3:#: Vnil
val l2= 1:#: 2 :#: Vnil
val l3= l1 vAdd l2
```

Output:

Compile error because you can't pairwise add vectors of different length

- In computer science and logic, a dependent type is a type whose definition depend on a value.
- Used to encode logic's qunatifiers like "for all" and "there exixts".
- Dependent types may help to reduce bugs

- A **function** has dependent type if the type of a function's results depends on the VALUE of its arguments

What does it mean to be "correct"?

Depends on the application domain,

- Functionally correct(e.g arithmetic operation on a CPU)
- Resourse safe (e.g runs within memory bound, no memory leaks, no deadlock….)
- Secure (e.g not allowing the access to another user's data)

TYPE:

Type – means of classifying the values

Example: value 94, "thing", [1,2,3]

- For machine, types describe how bit patterns in memory are to be interpreted
- For compiler or interpreter, types help ensure that bit patterns are interpreted consistently.

Comparison between general code and Pseudo-code

- **General Code**

```
float myDivide(float a, float b)
 { if (b == 0)
return ???;
Else
 return a / b;
 }
```

- **Dependent Type Code**

```
float myDivide3
(float a, float b, proof(b != 0)
    p)
{
return a / b;
 }
```

Auto Checking done here

Python Simple Example

```
from typing import Union
 def return_int_or_str(flag: bool) -> Union[str, int]:
if flag:
 return 'I am a string!‘
 return 0
```

Dependent Type

- » pip install mypy typing_extensions
- from typing import overload
- from typing_extension import Literal

Literal

Literal type represents a specific value of the specific type.

```
from typing_extensions import Literal
 def function(x: Literal[1]) -> Literal[1]:
 return x
function(1)
# => OK!
function(2)
 # => Argument has incompatible type "Literal[2]"; expected "Literal[1]"
```

Quantifiers

- A predicate becomes a proposition when we assign it fixed values. Predicate is true or false for all possible values in the universe or for some values in the universe.
- Quantification done in two ways: universal quantifier and existential quantifier

Universal Quantifier:

It is a symbol of logical representation, which specifies the statement within its range is true for every instance.

We use ∀ for universal quantifier

$$\forall x P(x)$$

Example:

P(x)= x must take a discrete mathematics course

Q(x)= x is a computer science student

“Every computer science student must take discrete mathematics course”

$$\forall x Q(x) \square P(x)$$

“Everybody must take a discrete mathematics course or be a computer science student”

$$\forall x (Q(x) V P(x))$$

Existential Quantifier

- Existential Quantifier of a predicate is the proposition " There exist an x in the universe of discourse such that P(x) is true.

Notation:

$\exists xP(x)$

which can be read "there exists an x".

It is denoted by the logical operator $\exists$, which resembles as inverted E

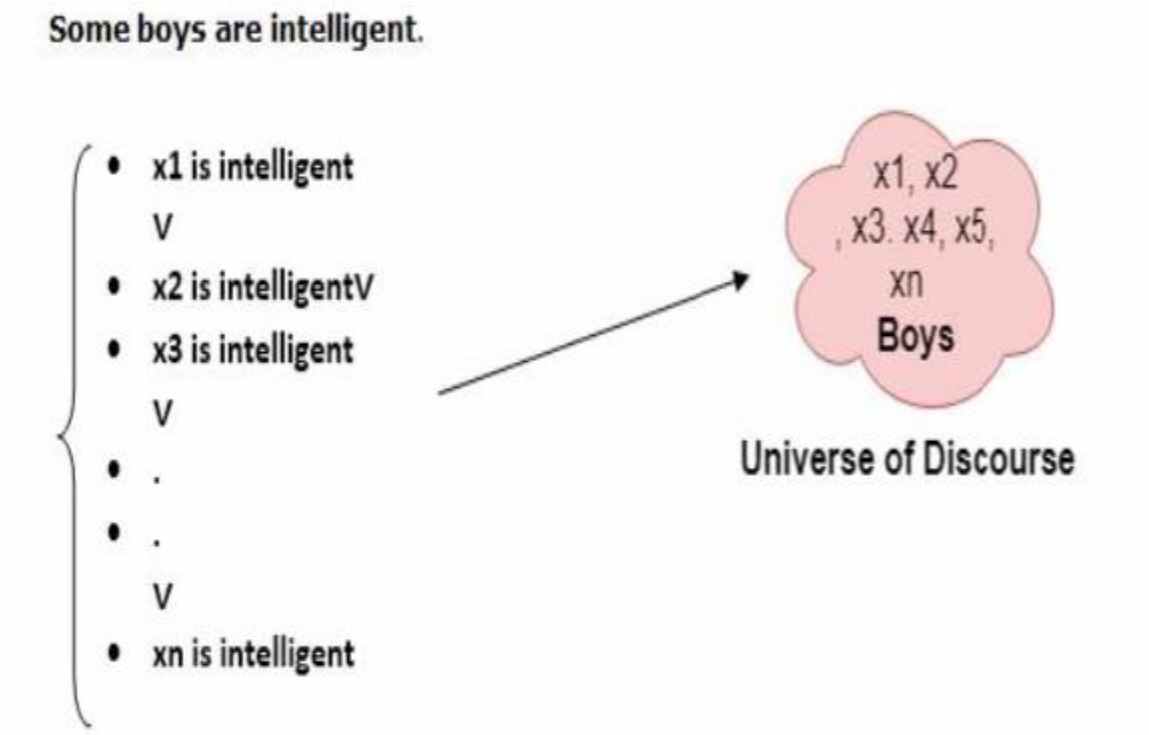
If x is a variable , then the existential quantifier will be $\exists x$ or $\exists(x)$

It can be read as:

there exists 'x'

for some 'x'

for atleast one 'x'



The mathematical notation is

$$\exists x: \text{boys}(x) \wedge \text{intelligent}(x)$$

Read as: there are some x where x is a boy who is intelligent

Example

1. All Birds fly.

   Here the predicate is fly(bird)

   $\forall x \text{bird}(x) \rightarrow \text{fly}(x)$

2. Every man respect his parents

   Here the predicate is respect(x,y) where x=man, y=parent

   $\forall x \text{man}(x) \rightarrow \text{respect}(x,\text{parent})$

3. Some boys plays cricket

   Here the predicate is play(x,y) where x=boys, y=game

   $\exists x \text{boys}(x) \rightarrow \text{play}(x,\text{cricket})$