Type Expressions for use in 6.001

Name	*Symbol*	*Example Usage*	*Expression whose type is example*
primitive	Sch*Name*	SchNum	`45`
		SchString	`“hello”`
function	→	SchNum → SchNum	`(lambda (x) (* x x))`
argument	,	SchNum, SchNum → SchNum	`+`
empty	ϕ	ϕ → SchNum	`(lambda () 6)`
variable	A, B, …	(A → B), A → B	`(lambda (f x) (f x))`
product	×	A, B → A × B	`cons`
abstract	*Name*	SchNum, SchNum → Rat	`make-rat`
special	AnyType	AnyType → SchBool[1]	`null?`
	Undefined	Symbol, AnyType → Undefined[2]	`define`

1. *Could also be written* A → SchBool *but more descriptive with AnyType*
2. *Could also be writen* Symbol, A → B *but more descriptive with Undefined.*

Precedence: A, B × C → D × E means (A, (B × C)) → (D × E).
Definitions: Can write Name = expression to define the name, then use in later expressions.

Wrinkles for advanced students

What to do about an infinite loop that never returns:

```
; f: SchNum → ϕ
(define f (lambda (x) (if (= x 0) (f x) (f x))))
```

Something that might be one of two (or multiple types):

```
; SchNum → SchBool | SchNull
(lambda (x) (if (= x 0) #t null))
```

Types can be recursive and parameterized:

list<A> = null | (A × list<A>)

This allows us to write the correct type for `map`:

(A→B), list<A> → list<B>