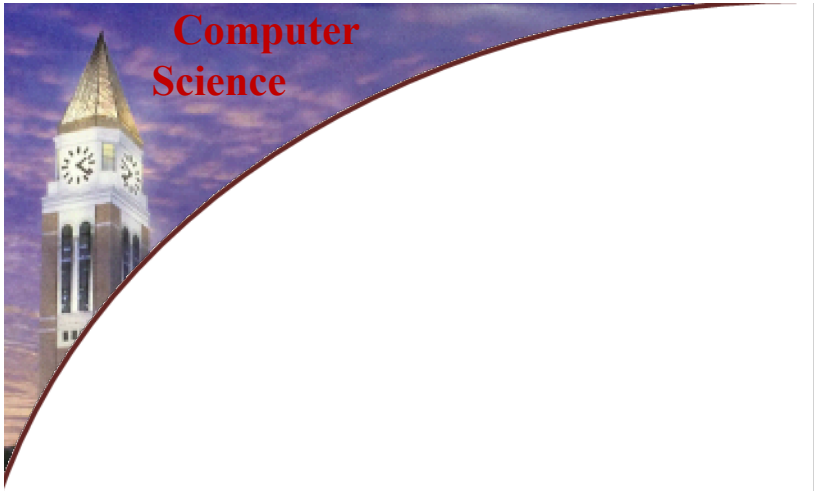
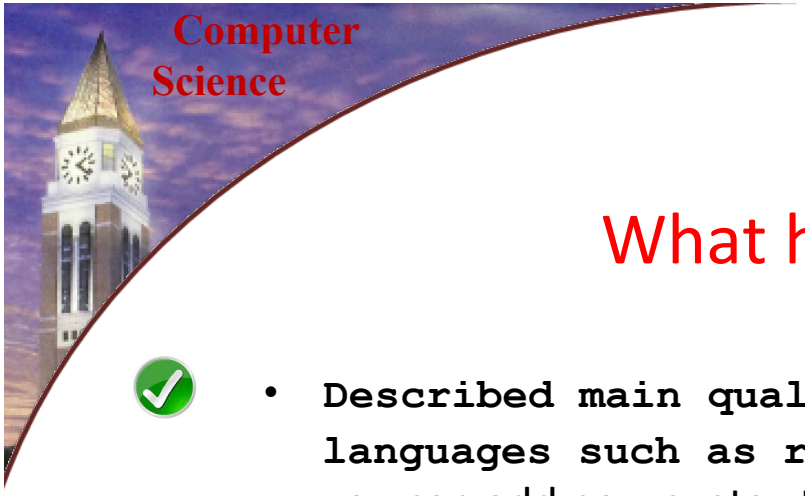


CSI 3350 Fall 2019
PROGRAMMING LANGUAGES

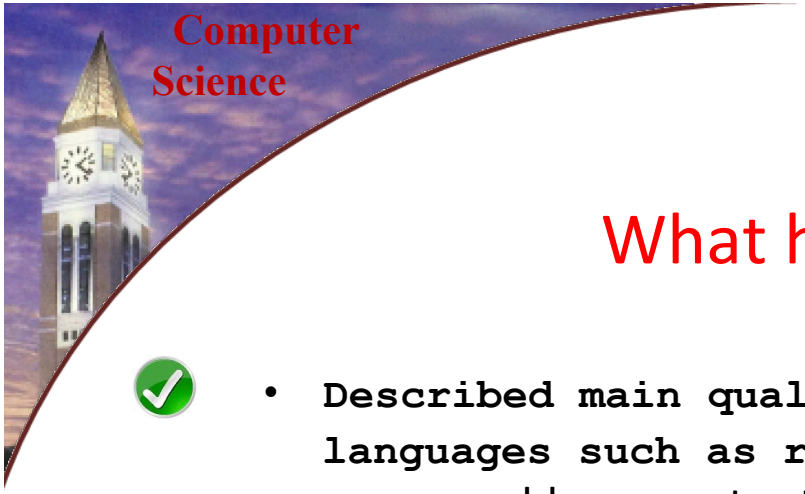


What have we covered so far ?

- **Described main quality criteria for high level programming languages such as readability, writability etc.** (for example you can add new syntax to the language to facilitate the readability of your program using *define-syntax-rule, define-syntax etc.*)
- **Described syntax of fundamental program components** (hw06 *loop, block structure, scoping mechanism, etc.*,)
- **Discussed fundamental concepts of operational semantics** (hw06, coding the **`value-of`** function)

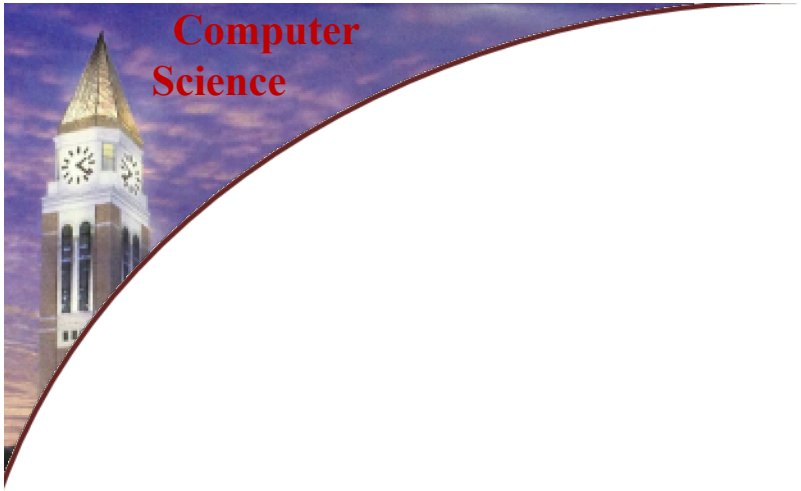
yet to cover

- Describe parameter passing and access to non-locals (hw07 – soon!)
- **Described data types and type systems** (*hw06, grammar, hw05, define-datatype*)
- **Apply major features of functional programming languages** (hw01~hw04, map, foldl, high order functions, lambda etc.)
- **Described activation records** (Sep 30 lecture notes, slides 43 ~ 66)



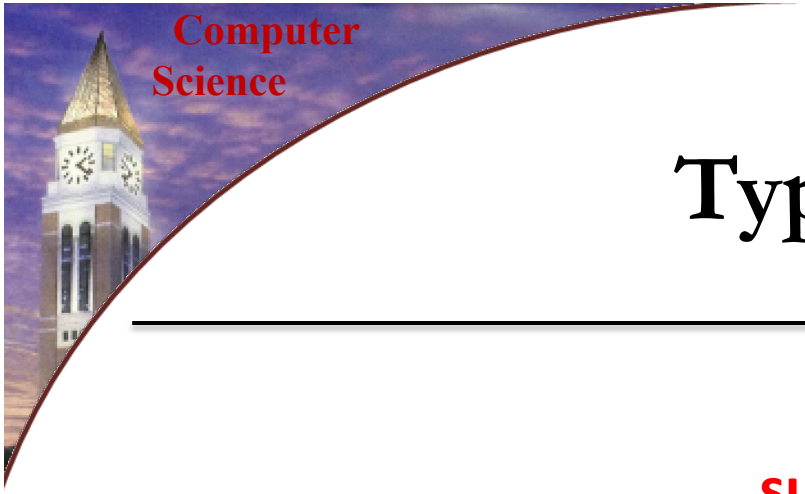
What have we covered so far ?

- **Described main quality criteria for high level programming languages such as readability, writability etc.** (for example you can add new syntax to the language to facilitate the readability of your program using *define-syntax-rule, define-syntax etc.*)
- **Described syntax of fundamental program components** (hw06 *loop, block structure, scoping mechanism, etc.*,)
- **Discussed fundamental concepts of operational semantics** (hw06, coding the **`value-of`** function)
- Describe parameter passing and access to non-locals (hw07)
- **Described data types and type systems** (*hw06, grammar, hw05, define-datatype*)
- **Apply major features of functional programming languages** (hw01~hw04, map, foldl, high order functions, lambda etc.)
- **Described activation records** (Sep 30 lecture notes, slides 43 ~ 66)



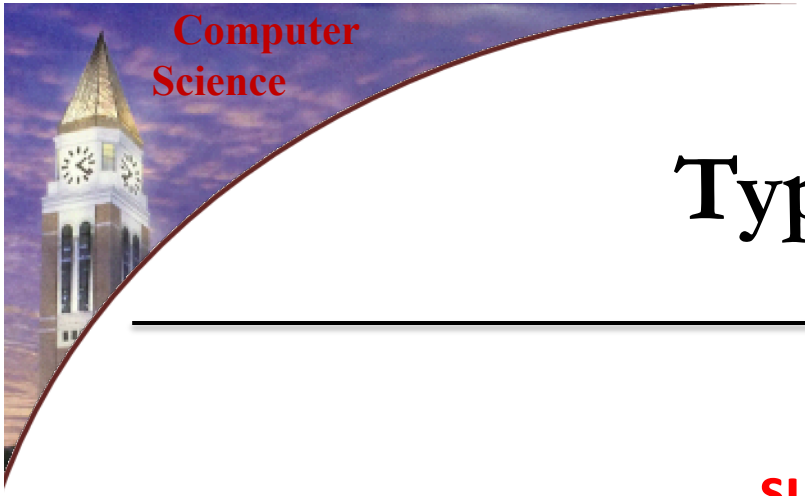
Type Inferences Through Typing Rules

(Types systems of programming languages)



Typing Values

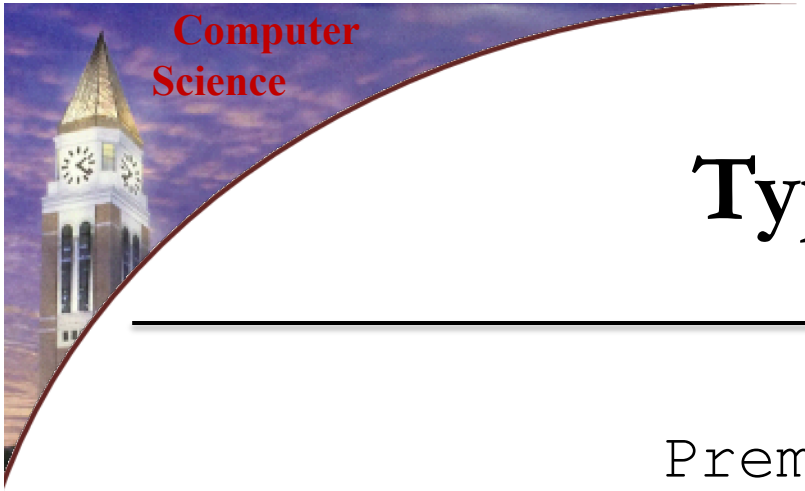
such as int, bool



Typing Values

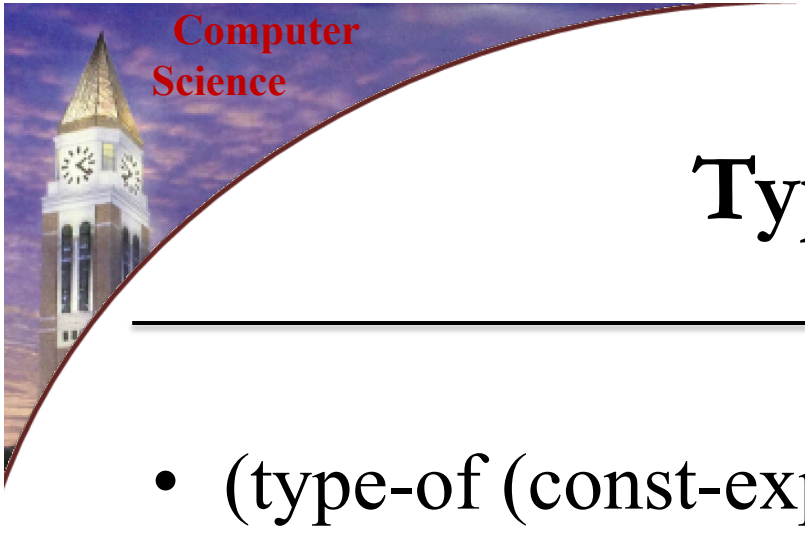
such as int, bool

Type environment - `tenv`



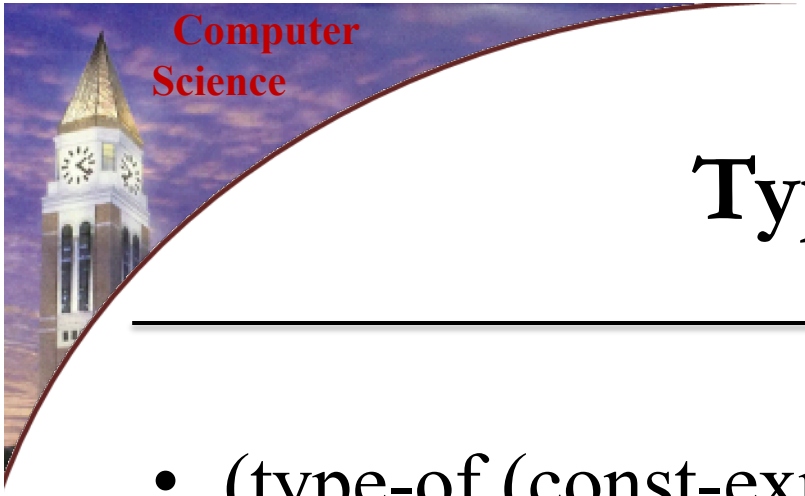
Typing Rules

$$\frac{\text{Premise(s)}}{\text{Conclusion}}$$



Typing Rules

- (type-of (const-exp num) tenv) = int



Typing Rules

- (type-of (const-exp num) tenv) = int
- (type-of (var-exp var) tenv) = (apply-tenv var)

Add Typing Rules For The LET Language

Program ::= *Expression*
`a-program (exp1)`

Expression ::= *Number*
`const-exp (num)`

Expression ::= `-(`*Expression* `,` *Expression*`)`
`diff-exp (exp1 exp2)`

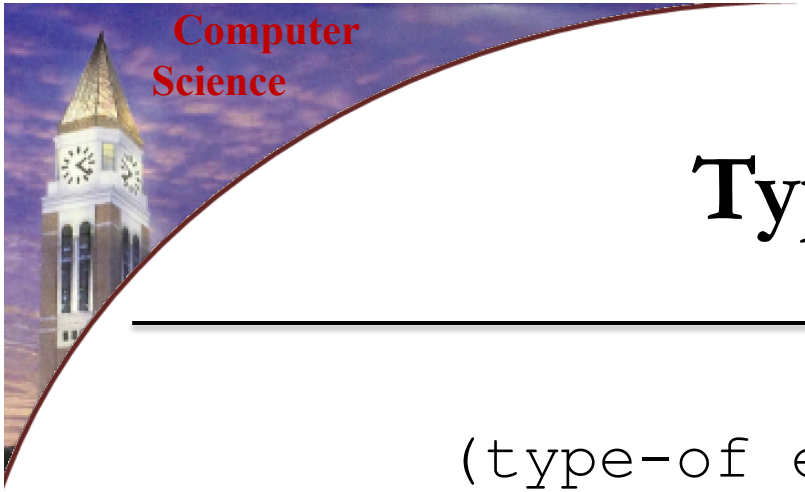
Expression ::= `zero? (`*Expression*`)`
`zero?-exp (exp1)`

Expression ::= `if` *Expression* `then` *Expression* `else` *Expression*
`if-exp (exp1 exp2 exp3)`

Expression ::= *Identifier*
`var-exp (var)`

Expression ::= `let` *Identifier* `=` *Expression* `in` *Expression*
`let-exp (var exp1 body)`

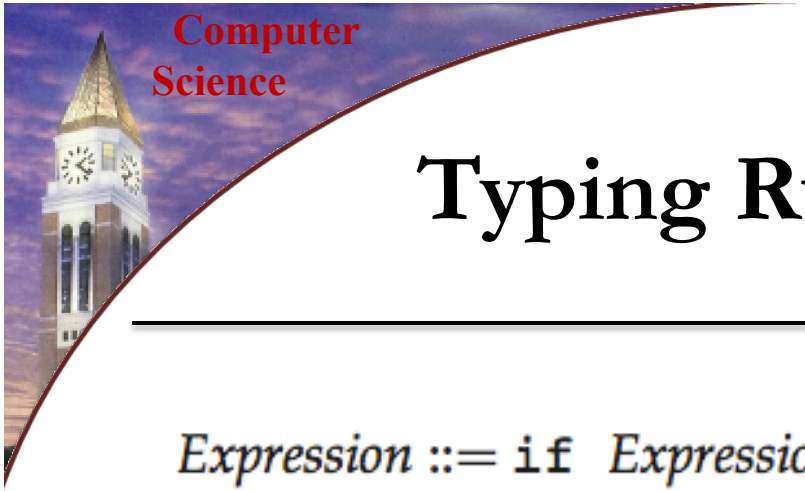
Figure 3.2 Syntax for the LET language



Typing Rules

$$\frac{\texttt{(type-of exp1 tenv) = int}}{\texttt{(type-of (zero?-exp exp1) tenv) = bool}}$$

$$\frac{\texttt{(type-of exp1 tenv) = int} \qquad \texttt{(type-of exp2 tenv) = int}}{\texttt{(type-of (diff-exp exp1 expr2) tenv) = int}}$$



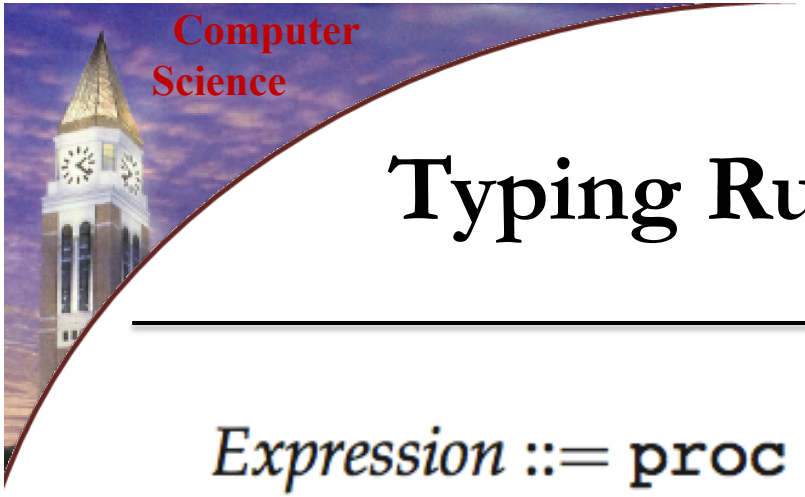
Typing Rule For `If-expr`

Expression ::= `if` *Expression* `then` *Expression* `else` *Expression*

`if-exp (exp1 exp2 exp3)`

```
(type-of exp1 tenv) = bool
(type-of exp2 tenv) = t
(type-of exp3 tenv) = t
```

```
(type-of (if-exp exp1 exp2 exp3) tenv) = t
```



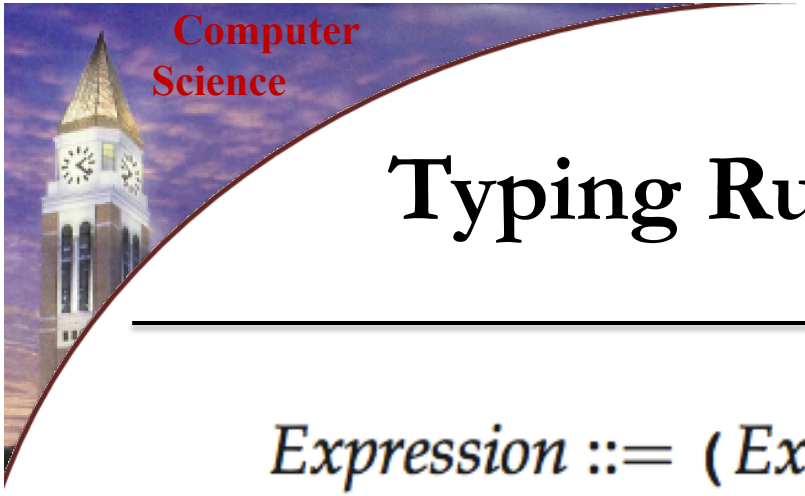
Typing Rule For `proc-exp`

Expression ::= `proc` (*Identifier*) *Expression*

`proc-exp (var body)`

```
(type-of body ([var t1] tenv)) = t2
```

```
(type-of (proc-exp var body) tenv) = t1->t2
```



Typing Rule For `call-exp`

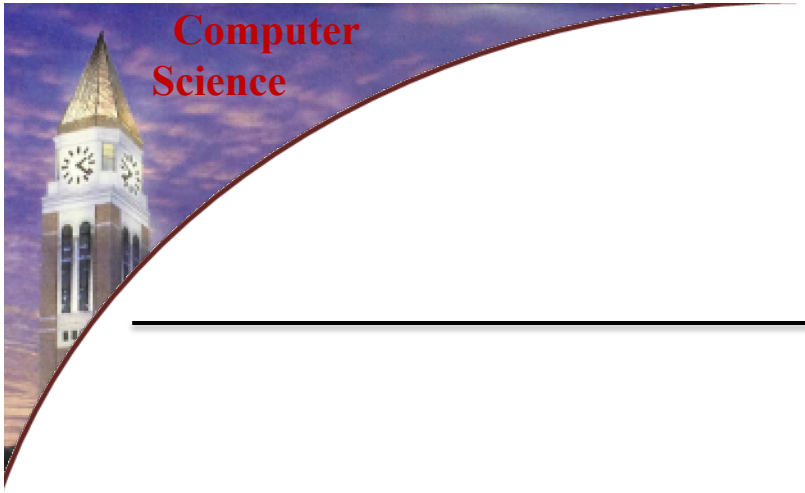
Expression ::= (*Expression* *Expression*)

```
call-exp (rator rand)
```

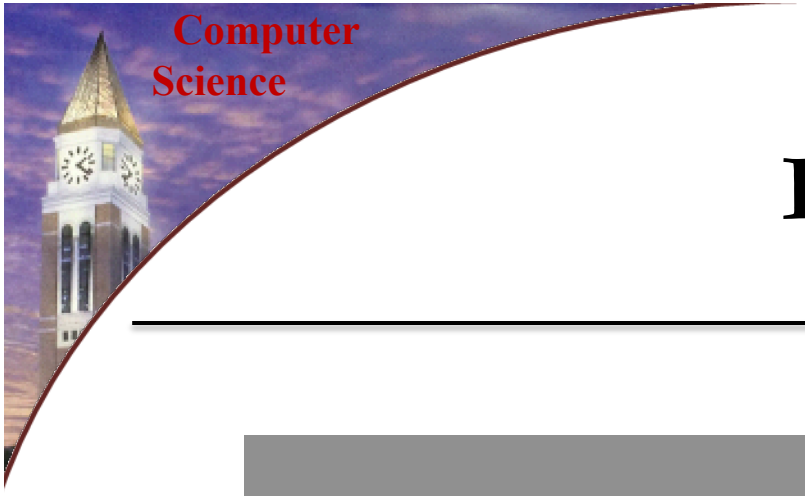
t1 -> t2

```
(type-of rator tenv) = ?
(type-of rand tenv) = ?t1
```

```
(type-of (call-exp rator rand) tenv) = t2
```



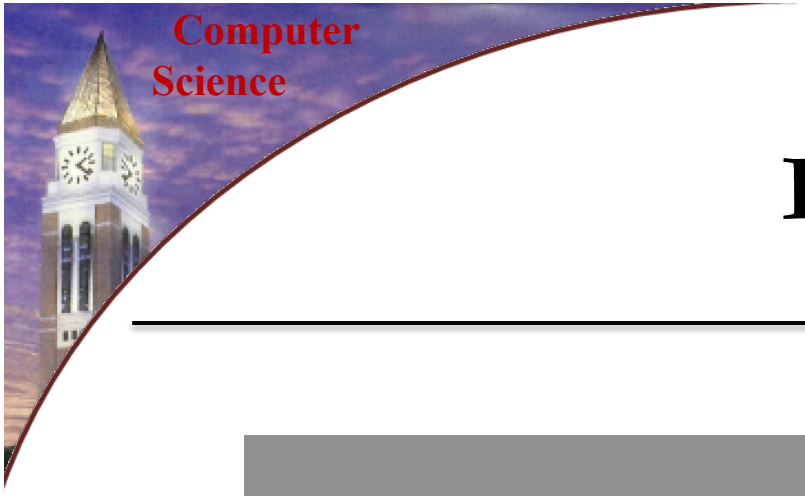
Type Inference



Example

What is the type of

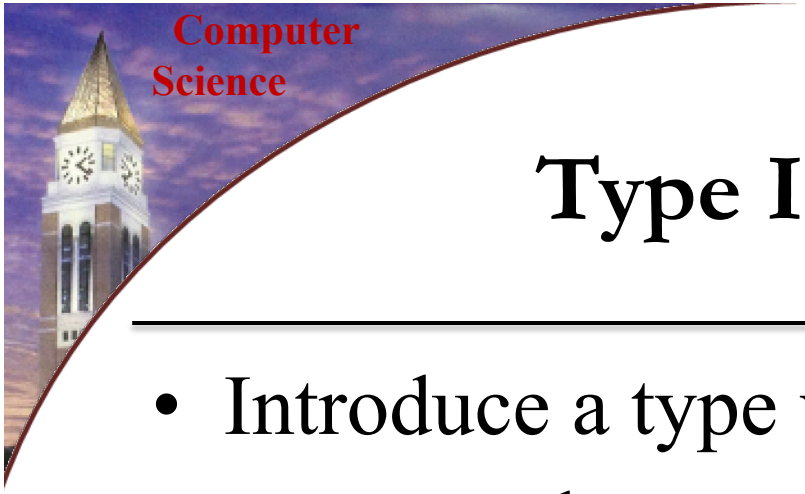
```
proc (f) proc (x)  -( (f 3) (f x) )
```



Example

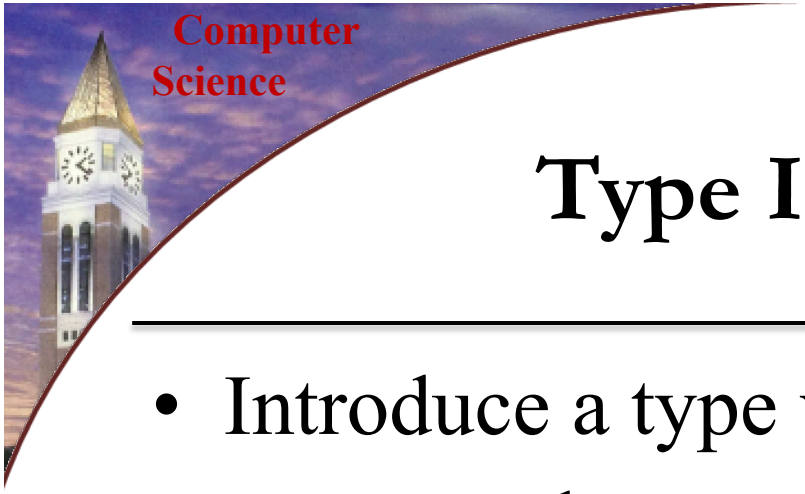
What is the type of

```
proc (f) proc (x)  -( (f 3) (f x) )
```



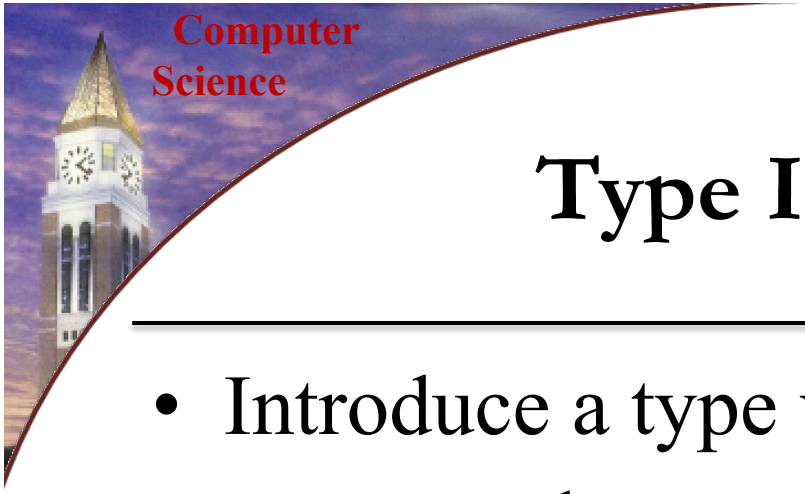
Type Inference Steps

- Introduce a type variable for:
  - every sub-expression, and
  - every bound variable



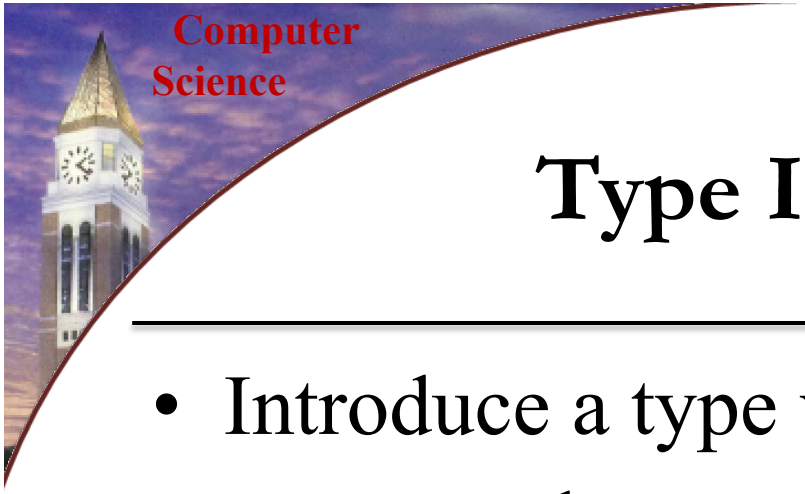
Type Inference Steps

- Introduce a type variable for:
  - every sub-expression, and
  - every bound variable
- Identify type constraints for each sub-expression
  - based on its typing rule



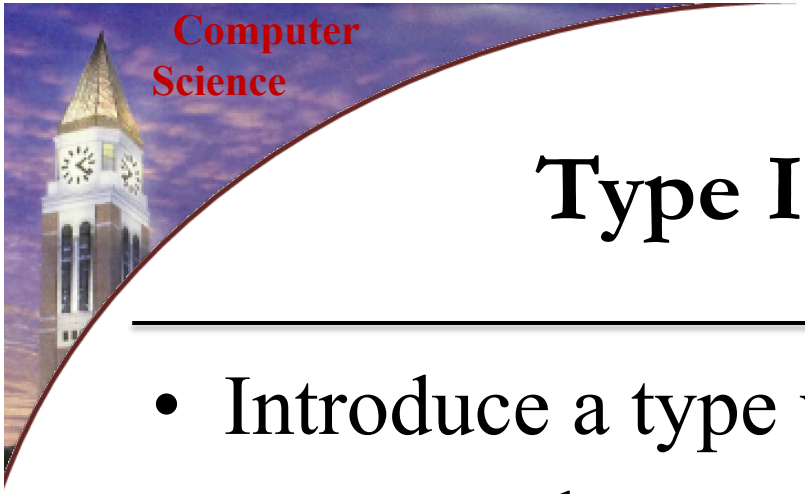
Type Inference Steps

- Introduce a type variable for:
  - every sub-expression, and
  - every bound variable
- Identify type constraints for each sub-expression
  - based on its typing rule
  - for example: `x` has to be a number for `3 + x`



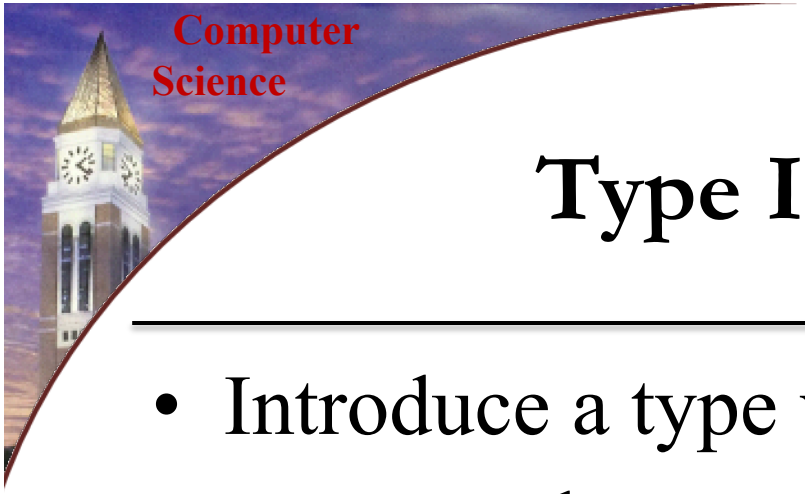
Type Inference Steps

- Introduce a type variable for:
  - every sub-expression, and
  - every bound variable
- Identify type constraints for each sub-expression
  - based on its typing rule
  - for example: `x` has to be a number for `3 + x`
- Solve the equations



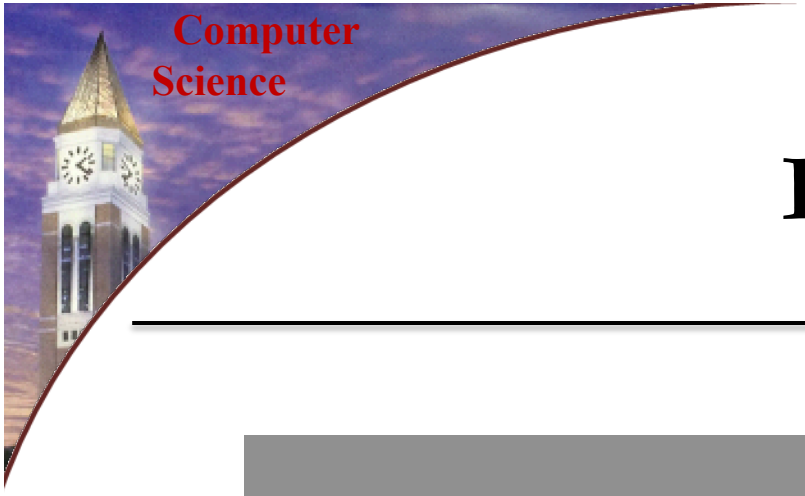
Type Inference Steps

- Introduce a type variable for:
  - every sub-expression, and
  - every bound variable
- Identify type constraints for each sub-expression
  - based on its typing rule
  - for example: `x` has to be a number for `3 + x`
- Solve the equations
  - using ***substitutions***



Type Inference Steps

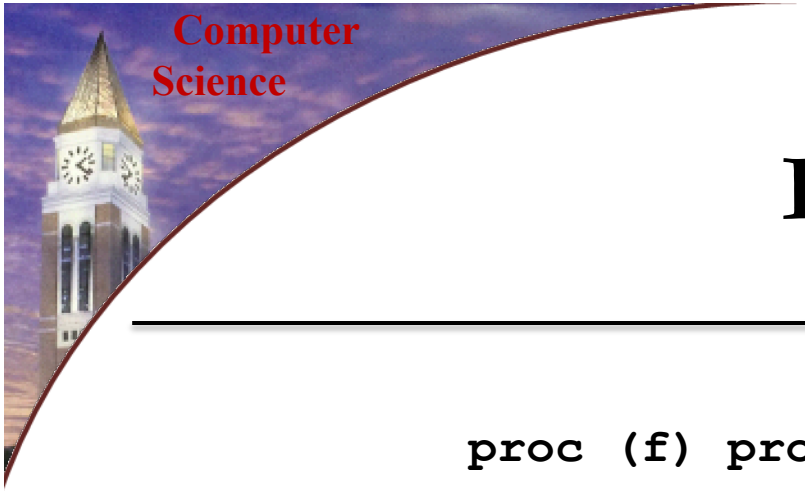
- Introduce a type variable for:
  - every sub-expression, and
  - every bound variable
- Identify type constraints for each sub-expression
  - based on its typing rule
  - for example: `x` has to be a number for `3 + x`
- Solve the equations (i.e., **unification**)
  - using ***substitutions***



Example

What is the type of

```
proc (f) proc (x)  -( (f 3) (f x) )
```



Example

```
proc (f) proc (x)  -( (f 3) (f x) )
```

(Sub)Expression	Type Variable
f	T_f
x	T_x
proc (f) proc(x) – ((f 3) (f x))	T_0
proc(x) – ((f 3) (f x))	T_1
– ((f 3) (f x))	T_2
(f 3)	T_3
(f x)	T_4

Example

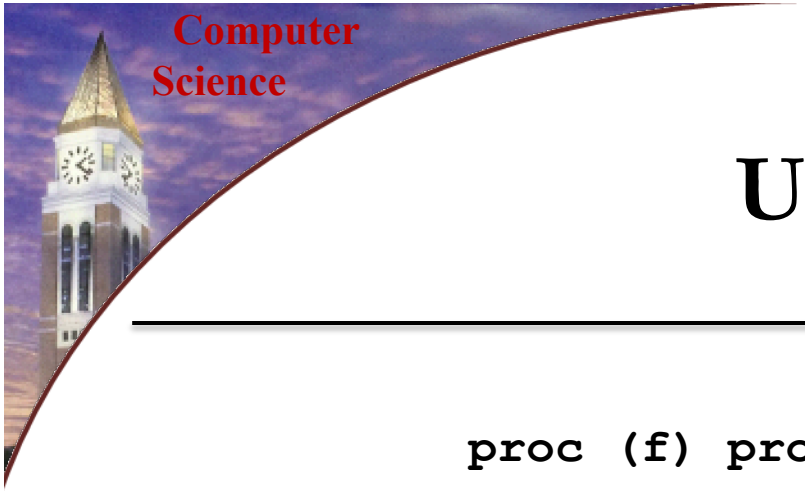
```
proc (f) proc (x)  -( (f 3) (f x) )
```

Expression	Type Variable	Equations
f	T_f	
x	T_x	
proc(f) proc(x) -((f 3) (f x))	T_0	$T_0 = T_f \rightarrow T_1$
proc(x) -((f 3) (f x))	T_1	$T_1 = T_x \rightarrow T_2$
-((f 3) (f x))	T_2	$T_3 = int$ $T_4 = int$ $T_2 = int$
(f 3)	T_3	$T_f = int \rightarrow T_3$
(f x)	T_4	$T_f = T_x \rightarrow T_4$

Example

```
proc (f) proc (x)  -( (f 3) (f x) )
```

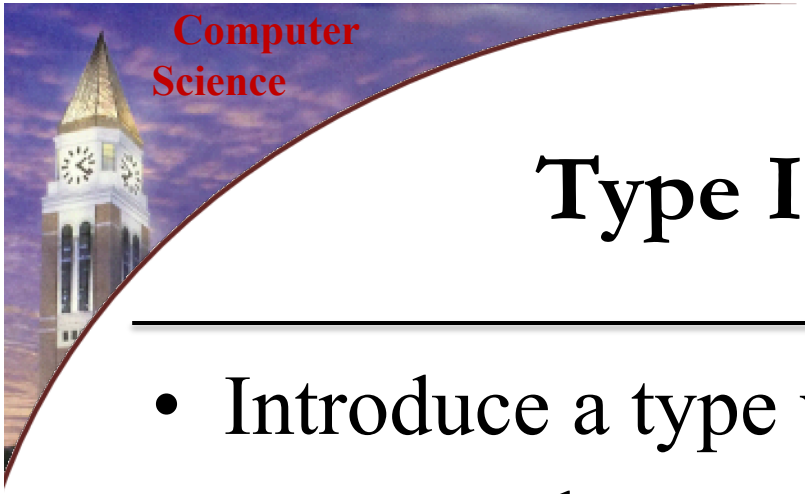
Expression	Type Variable	Equations
f	T_f	
x	T_x	
proc(f) proc(x) -((f 3) (f x))	T_0	$T_0 = T_f \rightarrow T_1$
proc(x) -((f 3) (f x))	T_1	$T_1 = T_x \rightarrow T_2$
-((f 3) (f x))	T_2	$T_3 = int$ $T_4 = int$ $T_2 = int$
(f 3)	T_3	$T_f = int \rightarrow T_3$
(f x)	T_4	$T_f = T_x \rightarrow T_4$



Unification

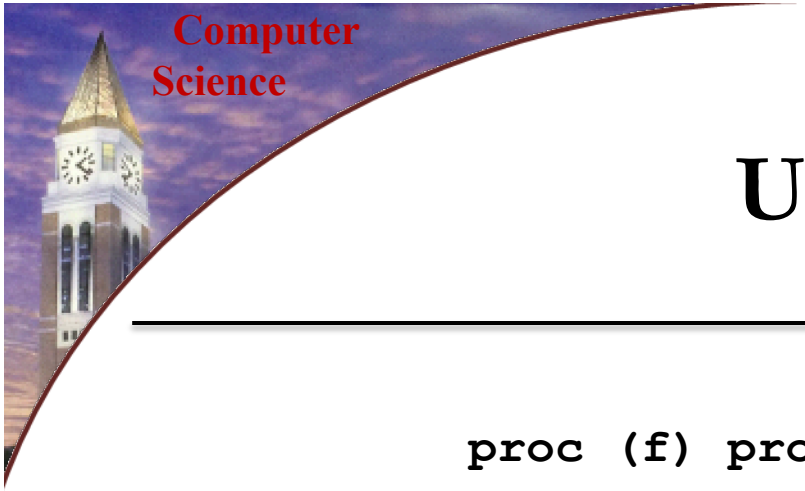
```
proc (f) proc (x)  -( (f 3) (f x) )
```

Equations	Substitutions
$T_0 = T_f \rightarrow T_1$	
$T_1 = T_x \rightarrow T_2$	
$T_3 = int$ $T_4 = int$ $T_2 = int$	
$T_f = int \rightarrow T_3$	
$T_f = T_x \rightarrow T_4$	



Type Inference Steps

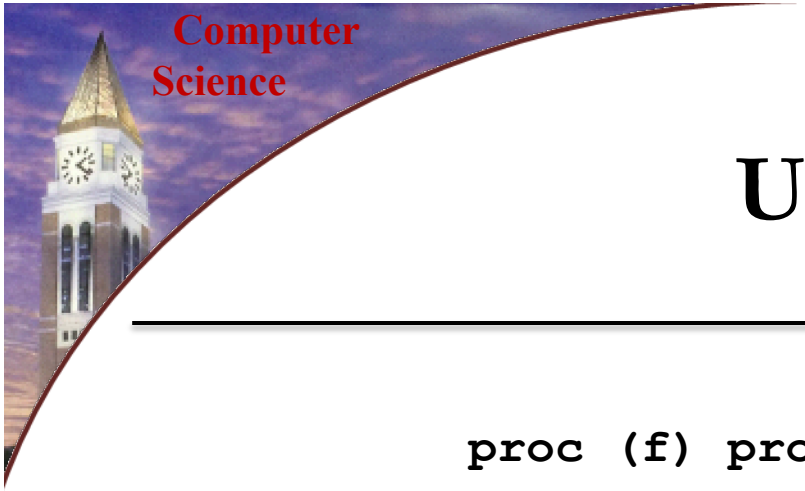
- Introduce a type variable for:
  - every sub-expression, and
  - every bound variable
- Identify type constraints for each sub-expression
  - based on its typing rule
  - for example: `x` has to be a number for `3 + x`
- Solve the equations (i.e., **unification**)
  - using ***substitutions***



Unification

```
proc (f) proc (x)  -( (f 3) (f x) )
```

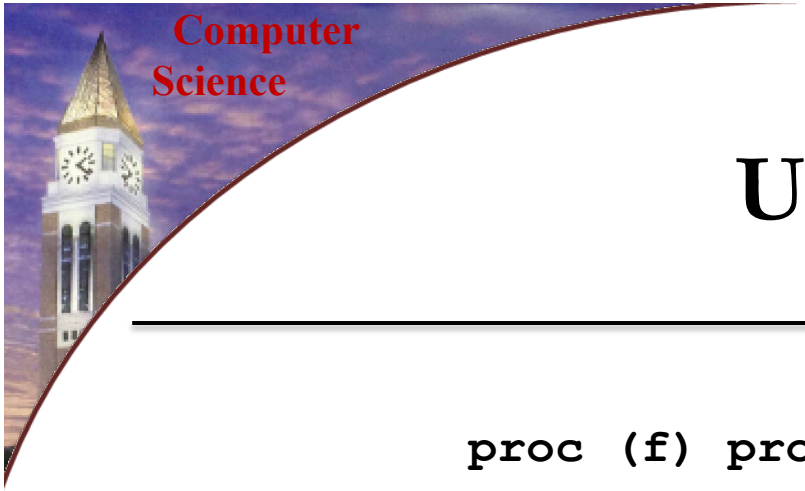
Equations	Substitutions
$T_0 = T_f \rightarrow T_1$	
$T_1 = T_x \rightarrow T_2$	
$T_3 = int$ $T_4 = int$ $T_2 = int$	
$T_f = int \rightarrow T_3$	
$T_f = T_x \rightarrow T_4$	



Unification

```
proc (f) proc (x)  -( (f 3) (f x) )
```

Equations	Substitutions
	$T_0 = T_f \rightarrow T_1$
$T_1 = T_x \rightarrow T_2$	
$T_3 = int$ $T_4 = int$ $T_2 = int$	
$T_f = int \rightarrow T_3$	
$T_f = T_x \rightarrow T_4$	



Unification

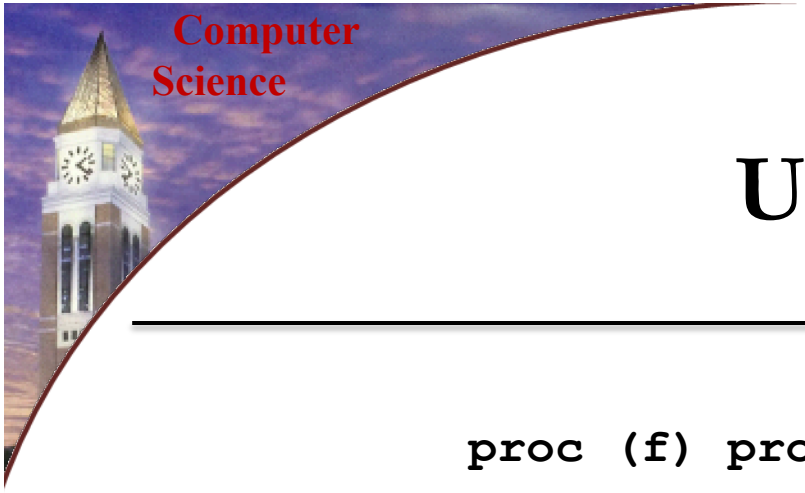
```
proc (f) proc (x)  -( (f 3) (f x) )
```

Equations	Substitutions
	$T_0 = T_f \rightarrow T_1$
	$T_1 = T_x \rightarrow T_2$
$T_3 = int$ $T_4 = int$ $T_2 = int$	
$T_f = int \rightarrow T_3$	
$T_f = T_x \rightarrow T_4$	

Unification

```
proc (f) proc (x)  -( (f 3) (f x) )
```

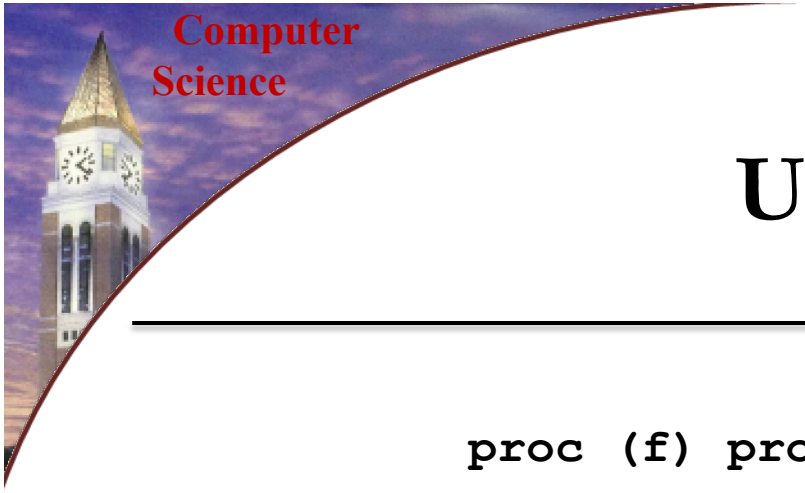
Equations	Substitutions
	$T_0 = T_f \rightarrow (T_x \rightarrow T_2)$
	$T_1 = T_x \rightarrow T_2$
$T_3 = int$ $T_4 = int$ $T_2 = int$	
$T_f = int \rightarrow T_3$	
$T_f = T_x \rightarrow T_4$	



Unification

```
proc (f) proc (x)  -( (f 3) (f x) )
```

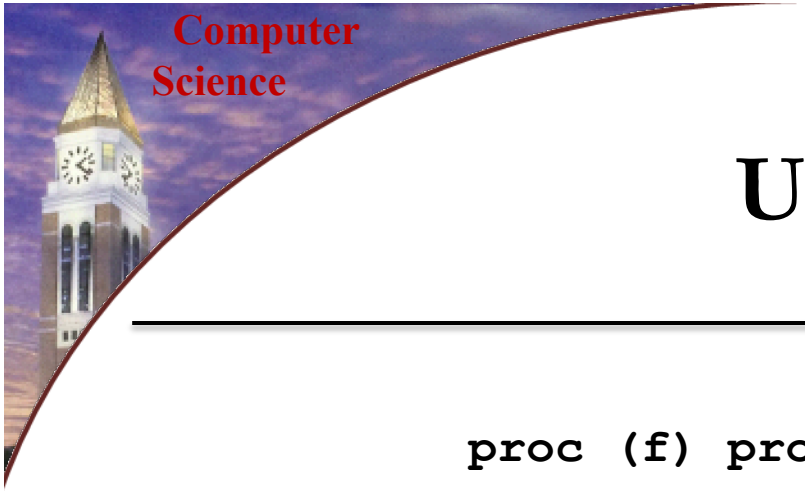
Equations	Substitutions
	$T_0 = T_f \text{ -> } (T_x \text{ -> } T_2)$
	$T_1 = T_x \text{ -> } T_2$
	$T_3 = \text{int}$ $T_4 = \text{int}$ $T_2 = \text{int}$
$T_f = \text{int -> } T_3$	
$T_f = T_x \text{ -> } T_4$	



Unification

```
proc (f) proc (x)  -( (f 3) (f x) )
```

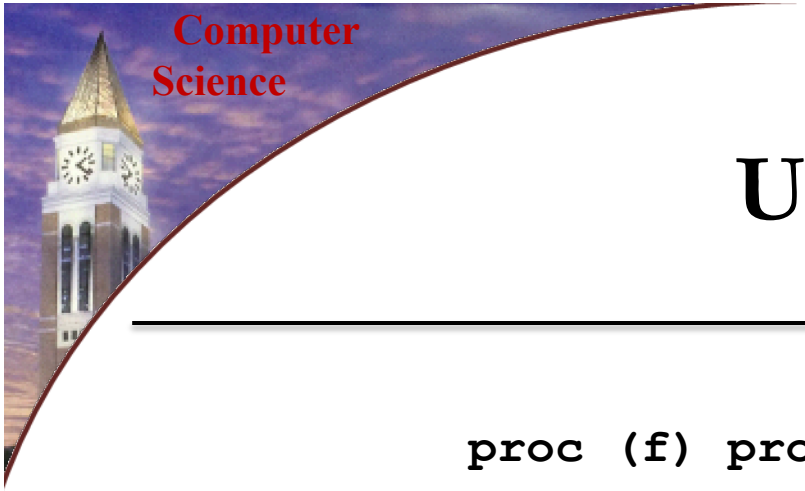
Equations	Substitutions
	$T_0 = T_f \text{ -> } (T_x \text{ -> } T_2)$
	$T_1 = T_x \text{ -> } T_2$
	$T_3 = \text{int}$ $T_4 = \text{int}$ $T_2 = \text{int}$
$T_f = \text{int -> } T_3$	
$T_f = T_x \text{ -> } T_4$	



Unification

```
proc (f) proc (x)  -( (f 3) (f x) )
```

Equations	Substitutions
	$T_0 = T_f \text{ -> } (T_x \text{ -> int})$
	$T_1 = T_x \text{ -> int}$
	$T_3 = \text{int}$ $T_4 = \text{int}$ $T_2 = \text{int}$
$T_f = \text{int -> } T_3$	
$T_f = T_x \text{ -> } T_4$	

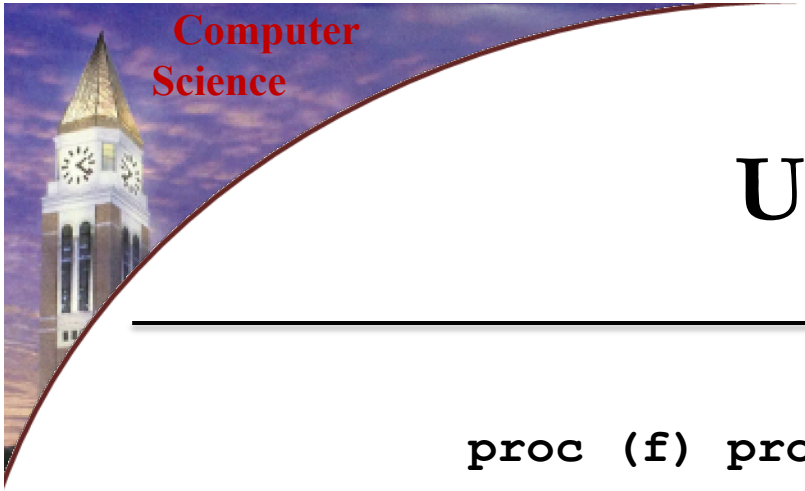


Unification

```
proc (f) proc (x)  -( (f 3) (f x) )
```

Equations	Substitutions
	$T_0 = T_f \rightarrow (T_x \rightarrow int)$
	$T_1 = T_x \rightarrow int$
	$T_3 = int$ $T_4 = int$ $T_2 = int$
$T_f = int \rightarrow T_3$	
$T_f = T_x \rightarrow T_4$	

next equation to move



Unification

```
proc (f) proc (x)  -( (f 3) (f x) )
```

Equations	Substitutions
	$T_0 = T_f \text{ -> } (T_x \text{ -> int })$
	$T_1 = T_x \text{ -> int}$
	$T_3 = \text{int}$ $T_4 = \text{int}$ $T_2 = \text{int}$
$T_f = \text{int -> } T_3$	
$T_f = T_x \text{ -> } T_4$	

Unification

```
proc (f) proc (x)  -( (f 3) (f x) )
```

Equations	Substitutions
	$T_0 = T_f \text{ -> } (T_x \text{ -> int })$
	$T_1 = T_x \text{ -> int}$
	$T_3 = \text{int}$ $T_4 = \text{int}$ $T_2 = \text{int}$
	$T_f = \text{int -> int}$
$T_f = T_x \text{ -> } T_4$	

Unification

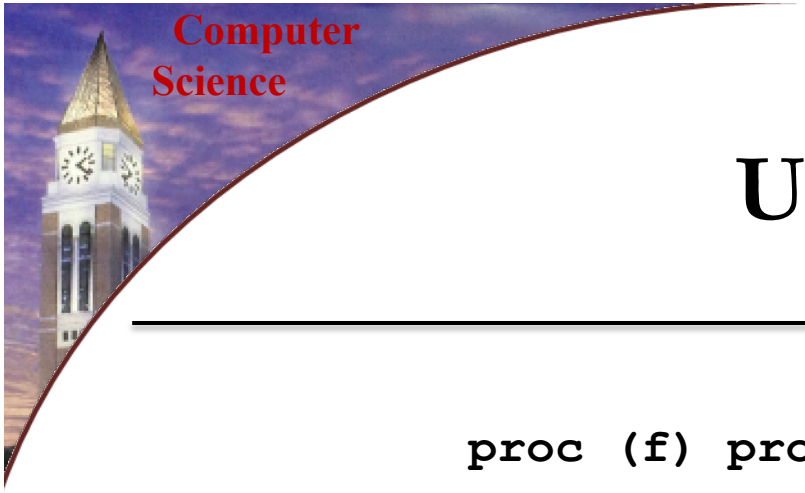
```
proc (f) proc (x)  -( (f 3) (f x) )
```

Equations	Substitutions
	T_0 = (int->int) -> (T_x -> int)
	T_1 = T_x -> int
	T_3 = int T_4 = int T_2 = int
	T_f = int -> int
T_f = T_x -> T_4	

Unification

```
proc (f) proc (x)  -( (f 3) (f x) )
```

Equations	Substitutions
	T_0 = (int->int) -> (T_x -> int)
	T_1 = T_x -> int
	T_3 = int T_4 = int T_2 = int
	T_f = int -> int
int -> int = T_x -> T_4	



Unification

```
proc (f) proc (x)  -( (f 3) (f x) )
```

Equations	Substitutions
	T_0 = (int->int) -> (T_x -> int)
	T_1 = T_x -> int
	T_3 = int T_4 = int T_2 = int
	T_f = int -> int
int -> int = T_x -> int	

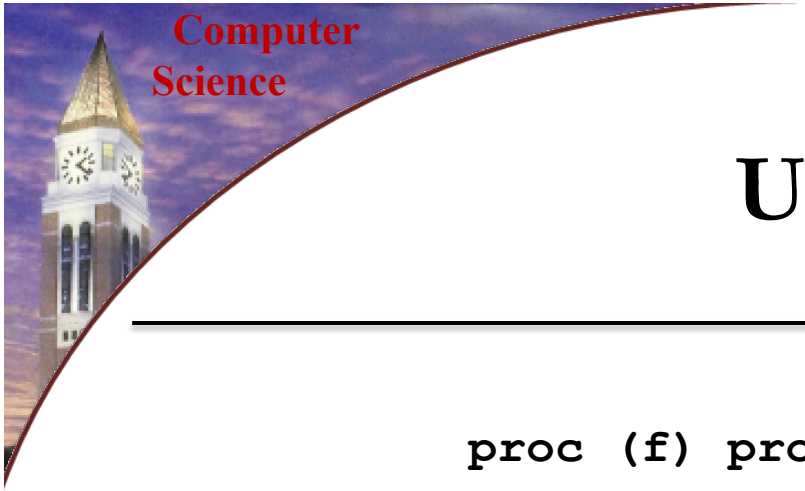
not a substitution yet

Solving this equation

Unification

```
proc (f) proc (x)  -( (f 3) (f x) )
```

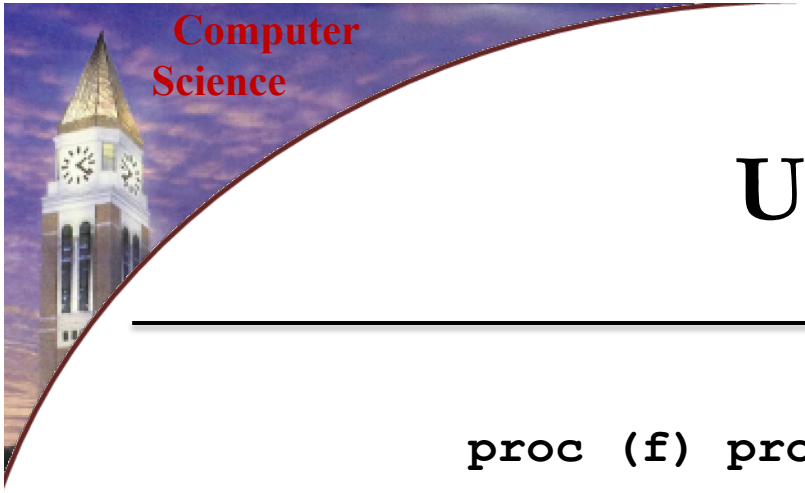
Equations	Substitutions
	T_0 = (int->int) -> (T_x -> int)
	T_1 = T_x -> int
	T_3 = int T_4 = int T_2 = int
	T_f = int -> int
	T_x = int



Unification

```
proc (f) proc (x)  -( (f 3) (f x) )
```

Equations	Substitutions
	T_0 = (int->int) -> (int -> int)
	T_1 = int -> int
	T_3 = int T_4 = int T_2 = int
	T_f = int -> int
	T_x = int



Unification

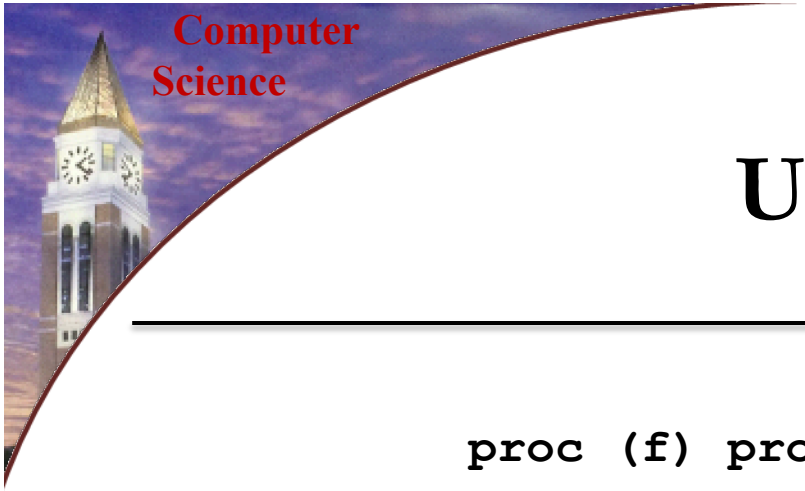
```
proc (f) proc (x)  -( (f 3) (f x) )
```

(Sub)Expression	Type Variable
f	T_f
x	T_x
proc (f) proc(x) – ((f 3) (f x))	T_0
proc(x) – ((f 3) (f x))	T_1
– ((f 3) (f x))	T_2
(f 3)	T_3
(f x)	T_4

Unification

```
proc (f) proc (x)  -( (f 3) (f x) )
```

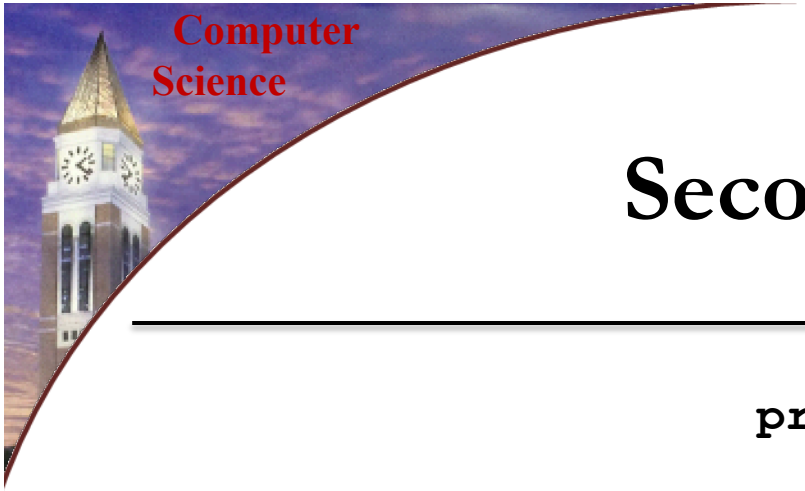
Equations	Substitutions
	T_0 = (int->int) -> (int -> int)
	T_1 = int -> int
	T_3 = int T_4 = int T_2 = int
	T_f = int -> int
	T_x = int



Unification

```
proc (f) proc (x)  -( (f 3) (f x) )
```

```
(int -> int) -> (int -> int)
```

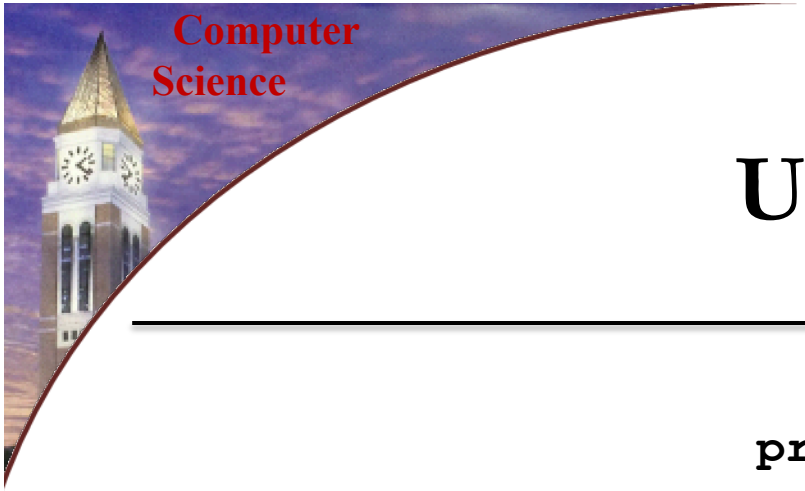


Second Example

```
proc (f) (f 11)
```

(Sub)Expression	Type Variable
f	T_f
proc (f) (f 11)	T_0
(f 11)	T_1

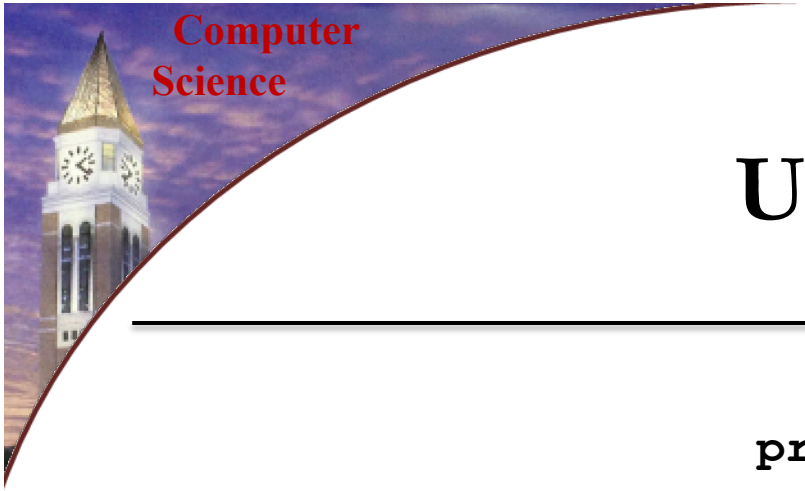
(Sub)Expression	Type Variable	Equations
f	T_f	
proc (f) (f 11)	T_0	$T_0 = T_f \rightarrow T_1$
(f 11)	T_1	$T_f = int \rightarrow T_1$



Unification

```
proc (f) (f 11)
```

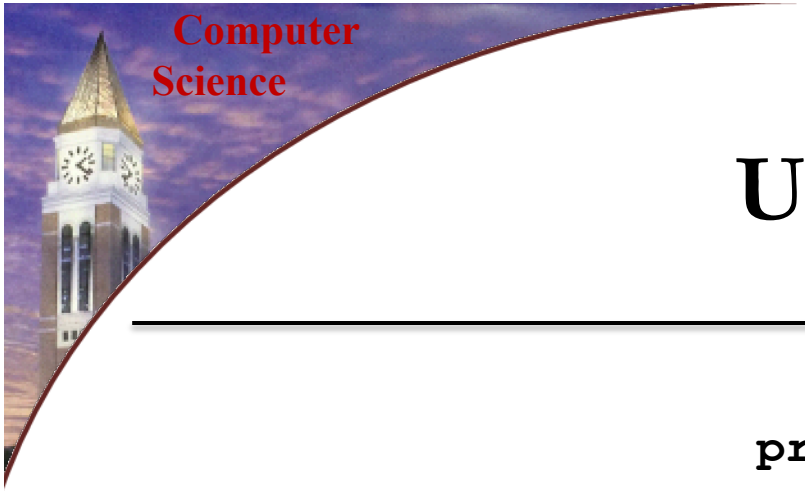
Euqations	Substitutions
$T_0 = T_f \rightarrow T_1$	
$T_f = int \rightarrow T_1$	



Unification

```
proc (f) (f 11)
```

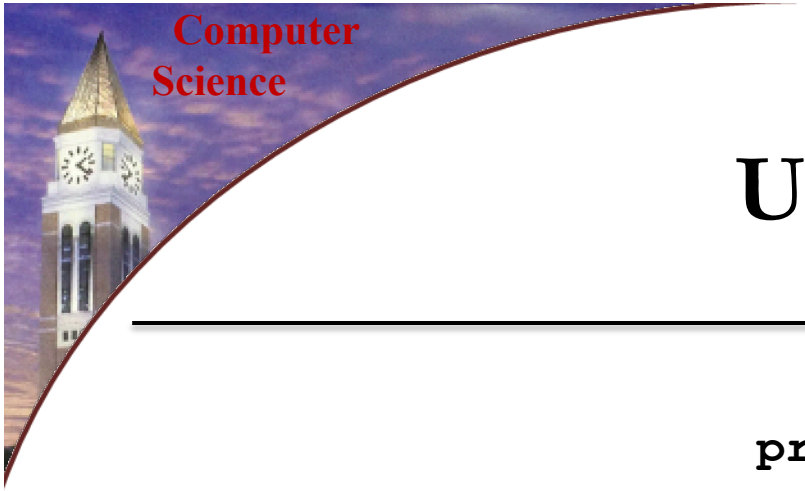
Euqations	Substitutions
	$T_0 = T_f \rightarrow T_1$
$T_f = int \rightarrow T_1$	



Unification

```
proc (f) (f 11)
```

Euqations	Substitutions
	$T_0 = T_f \rightarrow T_1$
	$T_f = int \rightarrow T_1$



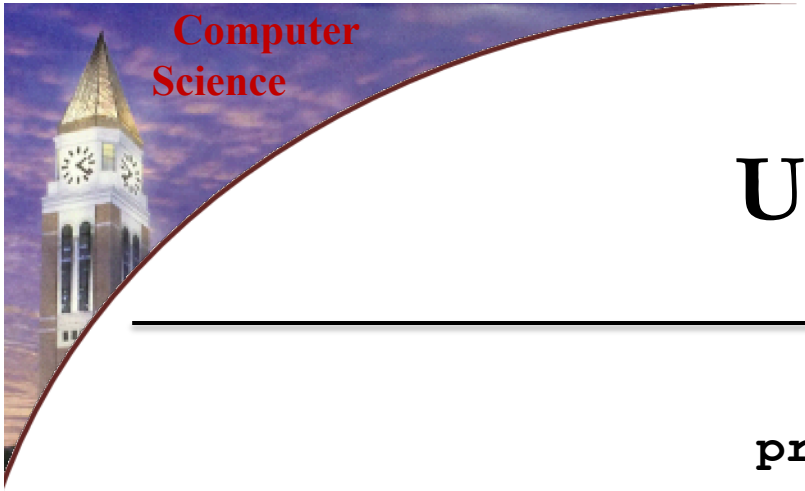
Unification

```
proc (f) (f 11)
```

Euqations	Substitutions
	T_0 = (int -> T_1) -> T_1
	T_f = int -> T_1

(Sub)Expression	Type Variable
f	T_f
proc (f) (f 11)	T_0
(f 11)	T_1

Are we done?

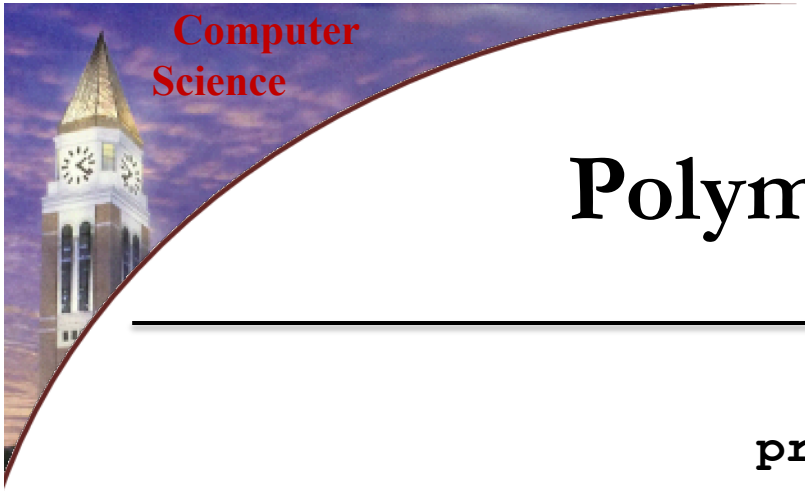


Unification

```
proc (f) (f 11)
```

$(int \rightarrow T_1) \rightarrow T_1$

Polymorphic in T_1, for any choice of T_1

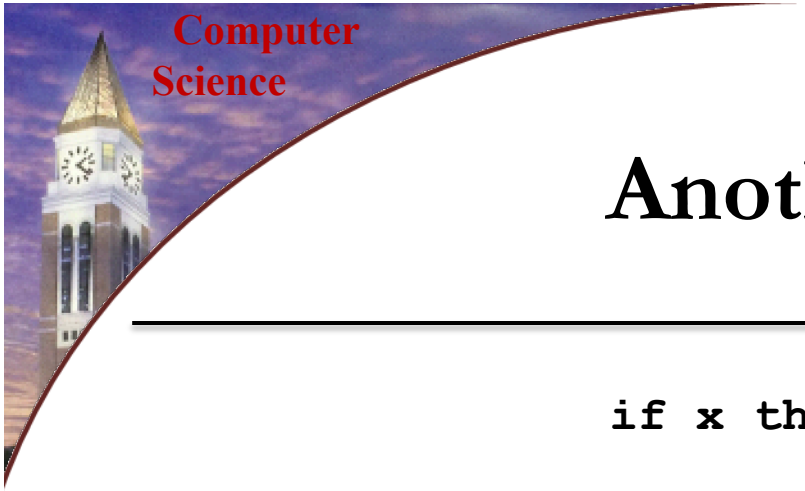


Polymorphic Types

```
proc (f) (f 11)
```

```
(int -> T1) -> T1
```

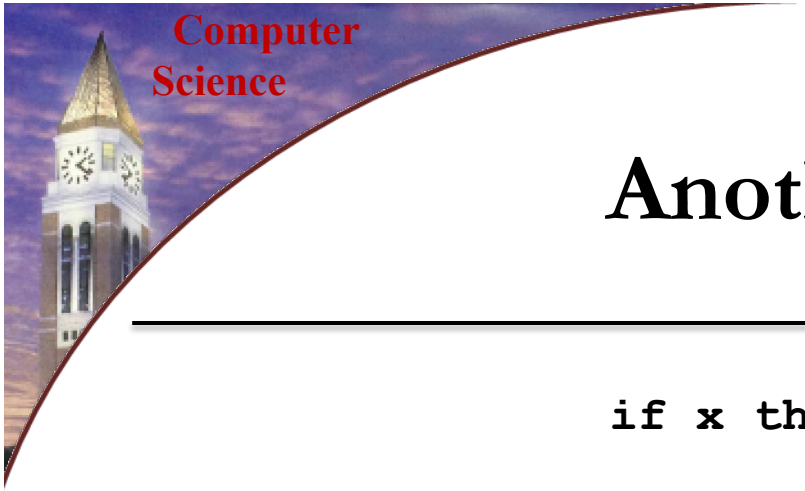
Polymorphic in T_1, for any choice of T_1



Another Example

```
if x then -(x,1) else 0
```

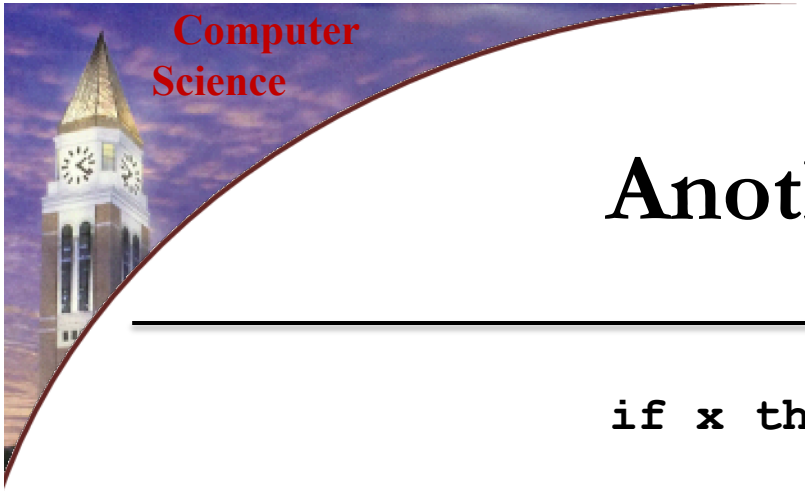
(Sub)Expressions	
x	
if x then -(x,1) else 0	
-(x,1)	



Another Example

```
if x then -(x,1) else 0
```

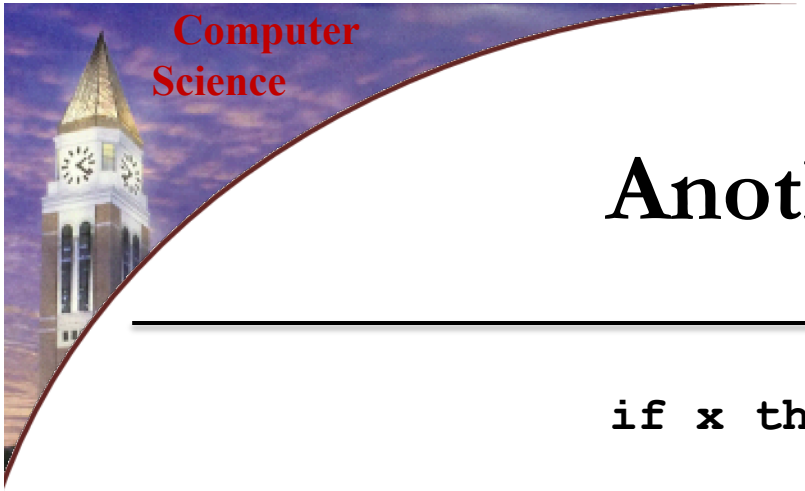
(Sub)Expressions	Type Variables
x	T_x
if x then -(x,1) else 0	T_0
-(x,1)	T_1



Another Example

```
if x then -(x,1) else 0
```

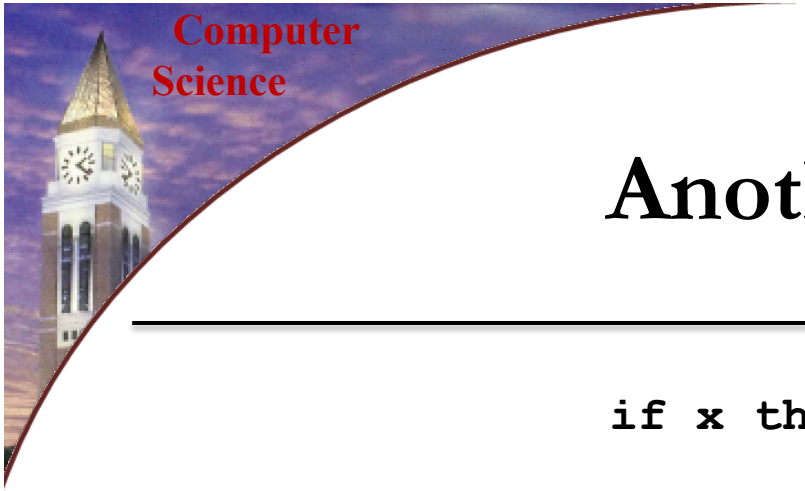
(Sub)Expression	Type Variable	Equations
x	T_x	
if x then -(x,1) else 0	T_0	
-(x,1)	T_1	



Another Example

```
if x then -(x,1) else 0
```

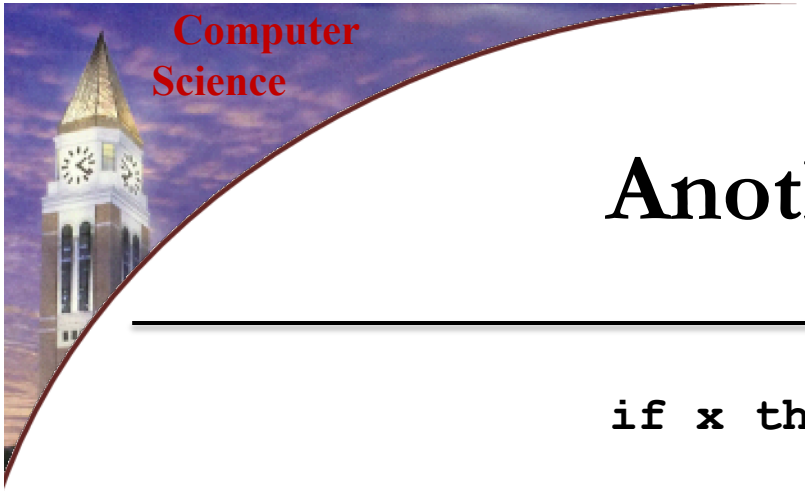
(Sub)Expression	Type Variable	Equations
x	T_x	
if x then -(x,1) else 0	T_0	T_x = bool $T_1 = T_0$ int = T_0
-(x,1)	T_1	



Another Example

```
if x then -(x,1) else 0
```

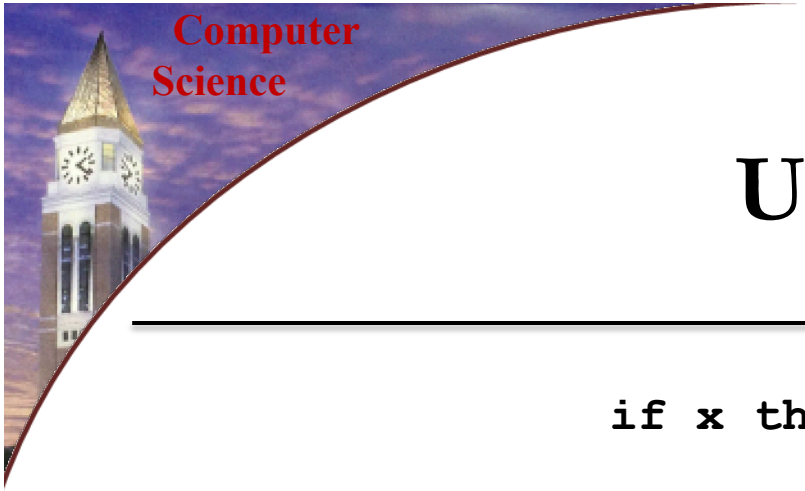
(Sub)Expression	Type Variable	Equations
x	T_x	
if x then -(x,1) else 0	T_0	$T_x = bool$ $T_1 = T_0$ $int = T_0$
-(x,1)	T_1	$T_x = int$ $T_1 = int$



Another Example

```
if x then -(x,1) else 0
```

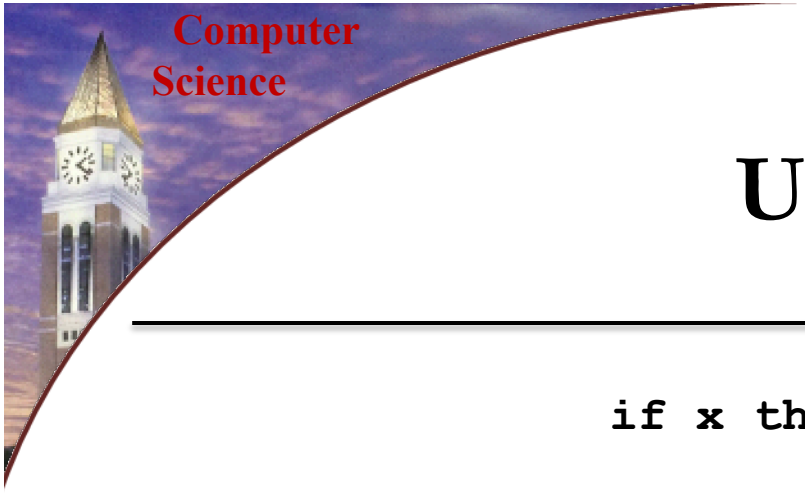
Equations	Substitutions
T_x = bool	
$T_1 = T_0$	
int = T_0	
T_x = int	
T_1 = int	



Unification

```
if x then -(x,1) else 0
```

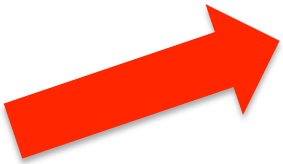
Equations	Substitutions
	T_x = bool
	$T_1 = T_0$
	T_0 = int
T_x = int	
T_1 = int	



Unification

```
if x then -(x,1) else 0
```

Equations	Substitutions
	T_x = bool
	$T_1 = T_0$
	T_0 = int
bool = int	
T_1 = int	



A contradiction!

Yet Another Example

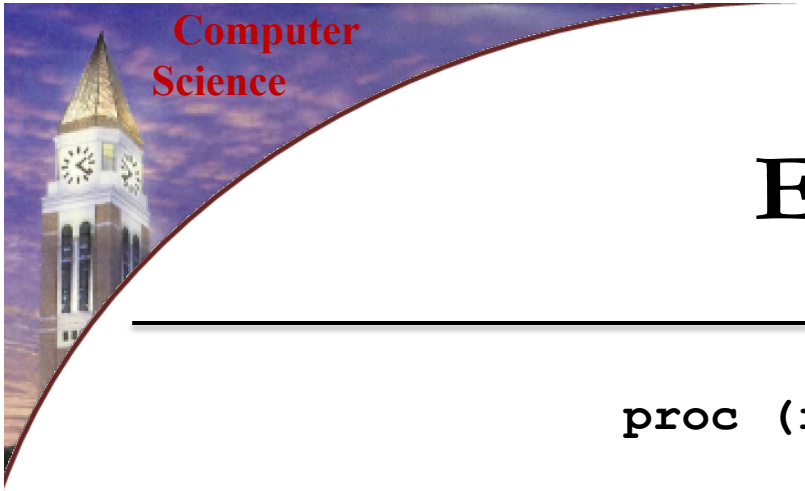
```
proc (f) zero? ((f f))
```

Expression	Type Variable

Yet Another Example

```
proc (f) zero? ((f f))
```

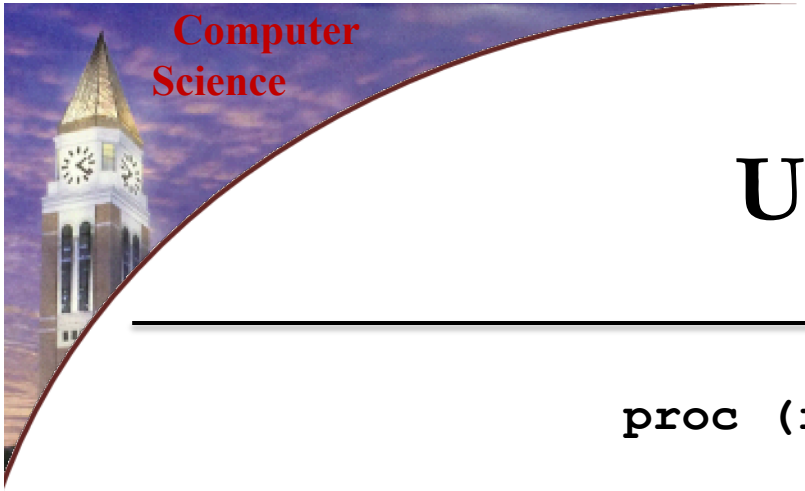
Expression	Type Variable
proc (f) zero? ((f f))	T_0
zero? ((f f))	T_1
(f f)	T_2
f	T_f



Equations

```
proc (f) zero? ((f f))
```

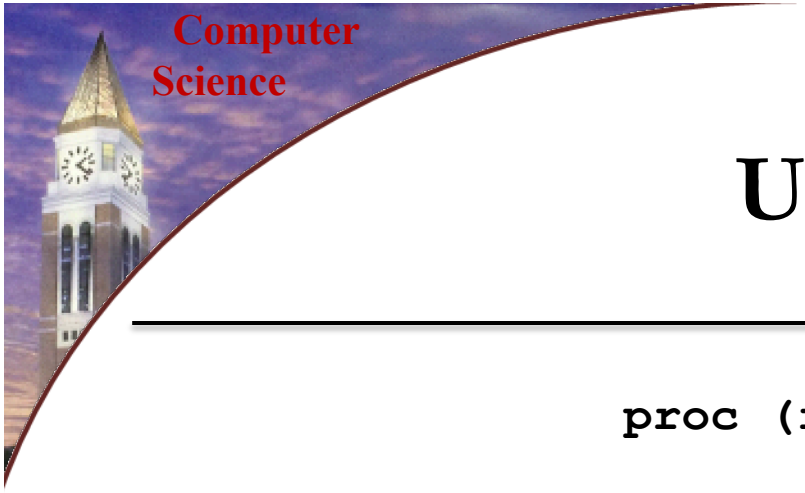
Expression	Type Variable	Equations
proc (f) zero? ((f f))	T_0	$T_0 = T_f \rightarrow T_1$
zero? ((f f))	T_1	$T_1 = bool$ $T_2 = int$
(f f)	T_2	$T_f = T_f \rightarrow T_2$



Unification

```
proc (f) zero? ((f f))
```

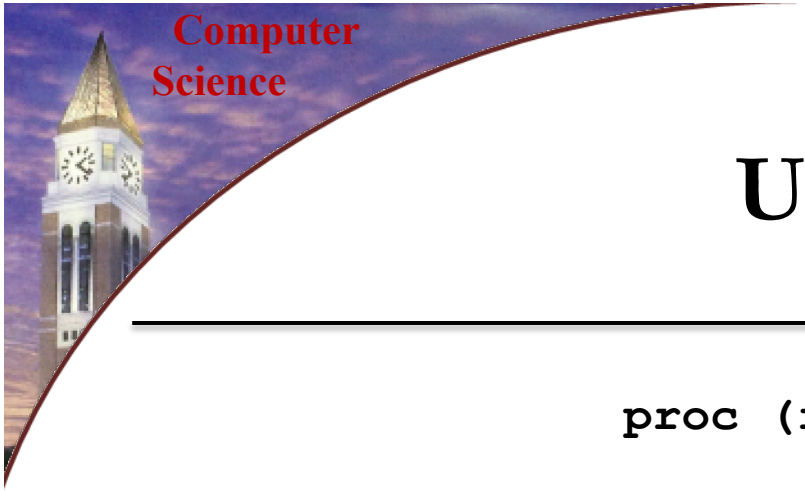
Equations	Substitutions
$T_0 = T_f \rightarrow T_1$	
$T_1 = \text{bool}$	
$T_2 = \text{int}$	
$T_f = T_f \rightarrow T_2$	



Unification

```
proc (f) zero? ((f f))
```

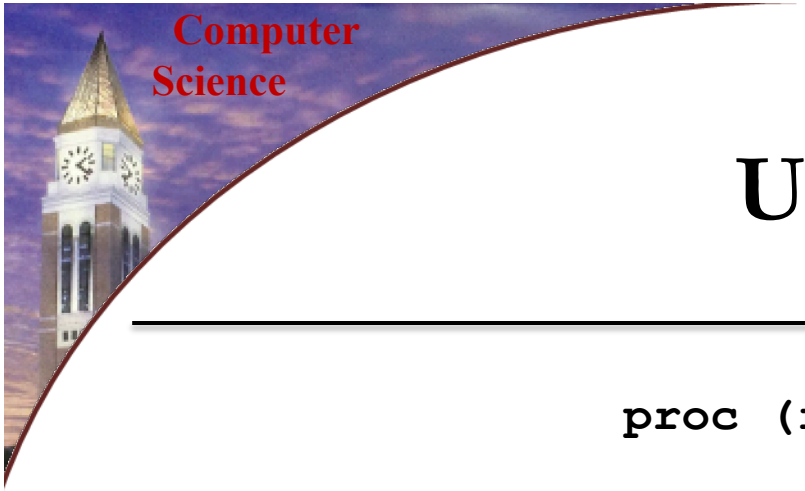
Equations	Substitutions
	$T_0 = T_f \rightarrow T_1$
	$T_1 = bool$
$T_2 = int$	
$T_f = T_f \rightarrow T_2$	



Unification

```
proc (f) zero? ((f f))
```

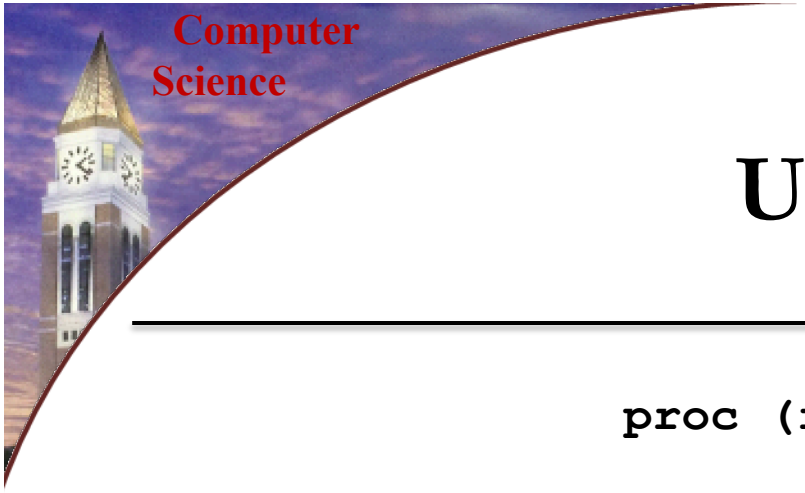
Equations	Substitutions
	$T_0 = T_f$ -> bool
	T_1 = bool
T_2 = int	
$T_f = T_f$ -> T_2	



Unification

```
proc (f) zero? ((f f))
```

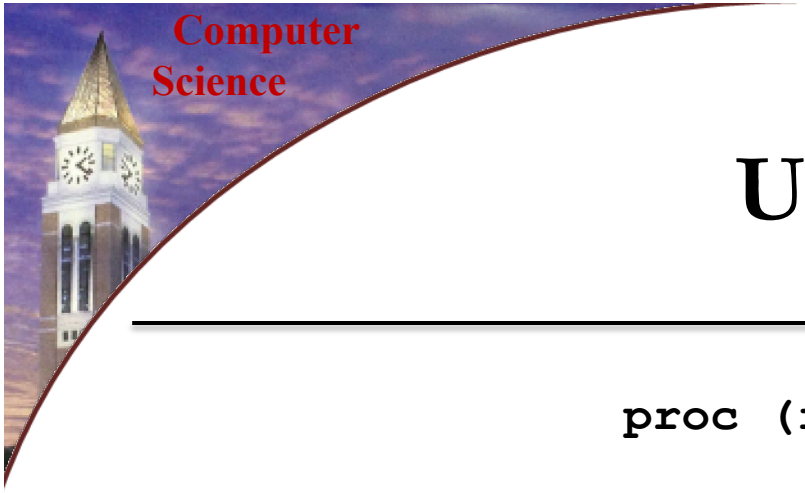
Equations	Substitutions
	$T_0 = T_f$ -> bool
	T_1 = bool
	T_2 = int
$T_f = T_f$ -> T_2	



Unification

```
proc (f) zero? ((f f))
```

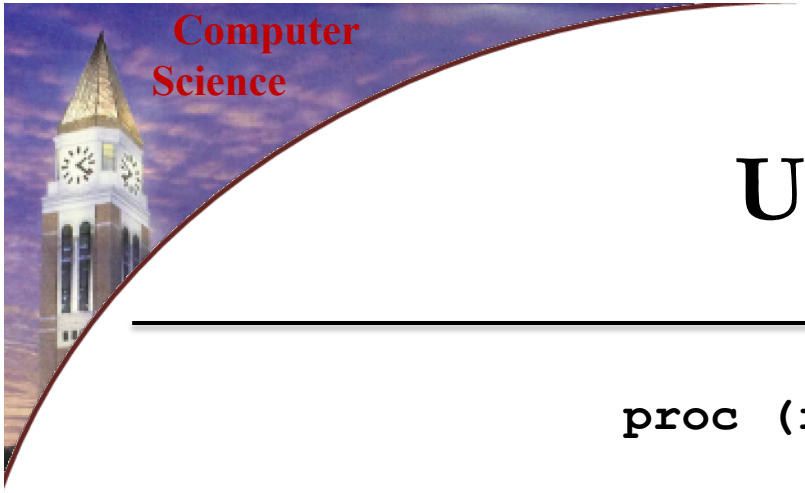
Equations	Substitutions
	$T_0 = T_f$ -> bool
	T_1 = bool
	T_2 = int
$T_f = T_f$ -> int	



Unification

```
proc (f) zero? ((f f))
```

Equations	Substitutions
	$T_0 = T_f$ -> bool
	T_1 = bool
	T_2 = int
	$T_f = T_f$ -> int

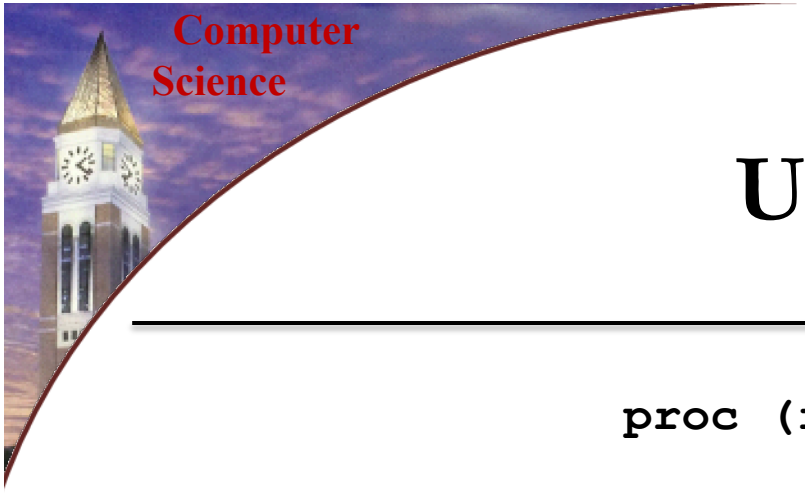


Unification

```
proc (f) zero? ((f f))
```

Equations	Substitutions
	$T_0 = T_f$ -> bool
	T_1 = bool
	T_2 = int
	$T_f = T_f$ -> int

Turing
Undecidable !



Unification

```
proc (f) zero? ((f f))
```

Equations	Substitutions
	$T_0 = T_f$ -> bool
	T_1 = bool
	T_2 = int
	$T_f = T_f$ -> int

Turing Undecidable !

Unsolvable by any computer !

CSI 3350 Fall 2019
PROGRAMMING LANGUAGES

CSI 3350 Fall 2019
PROGRAMMING LANGUAGES

Thank you !