Bounded Lattice Type System

Bidirectional type-directed program execution

Robert Smart

Overview

- BLTS is the type system of the experimental wombat language
  - https://github.com/rks987/marsupial (wombat is a marsupial library)
  - that code implements a small illustrative subset
- The subtyping hierarchy
- Relevant aspects of Wombat
- Procedures in the hierarchy
- The execution engine: type driven execution

What are types?

Mathematical Foundations folk have claimed the notion of “type”:

> “an element of one type can never be an element of another type”
> – Mike Shulman

- Not English usage: a dog is also a mammal.
- Not even the way most Mathematicians think
  - An integer is also a rational
  - A group is also a monoid

BLTS tries to push the idea of a hierarchy of types to the limit.

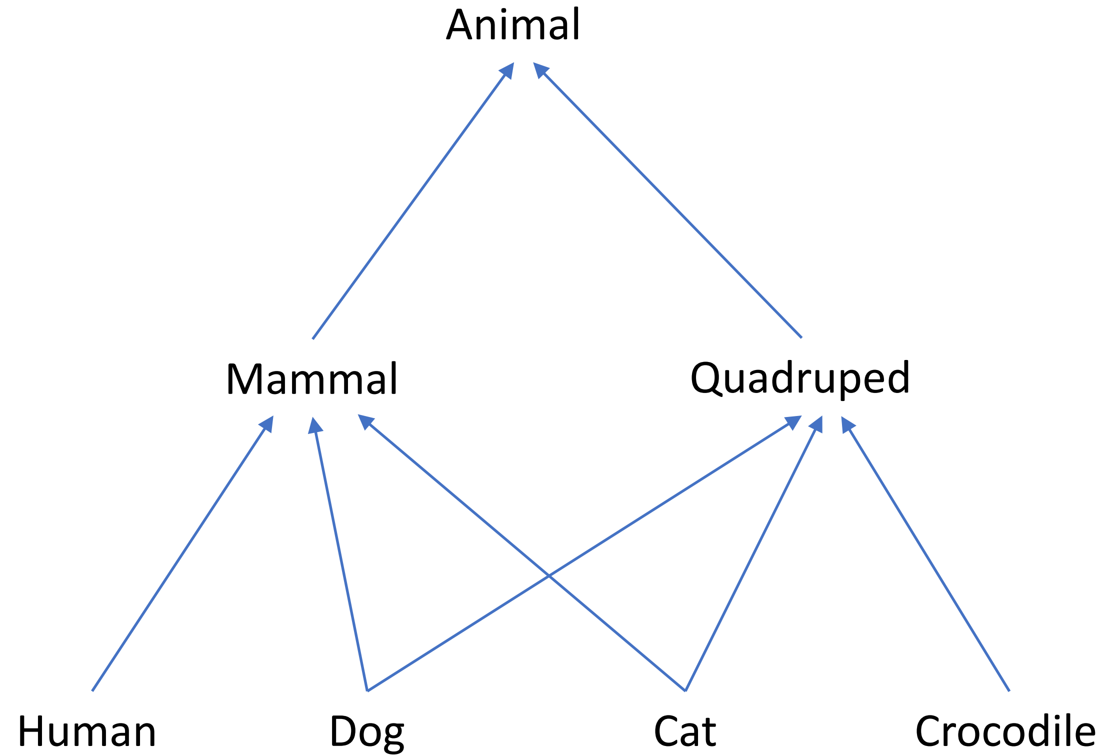
Subtyping

`Dog` **isA** `Mammal`

`Quadruped` **isA** `Animal`

etc, etc

- Properties go down as you go up
- Diamonds must commute
  - Prove or run-time check.
- Inverse pair of procedures:
  - Up is 1-1 total, always succeeds.
  - Down fails when input is not of that subtype.
  - Inverses of each other where both defined.



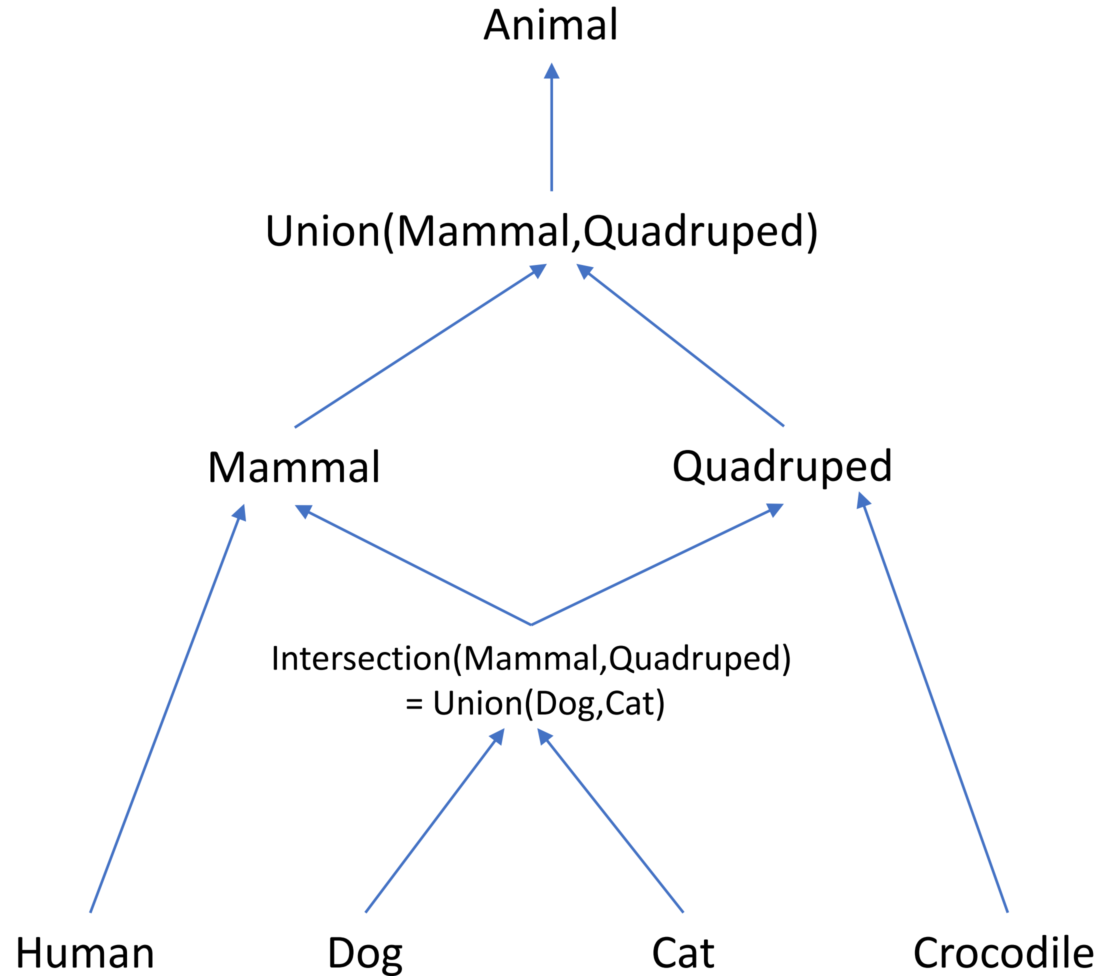
Lattice (order)

Least Upper Bound (lub / join / sup)

- Union in wombat

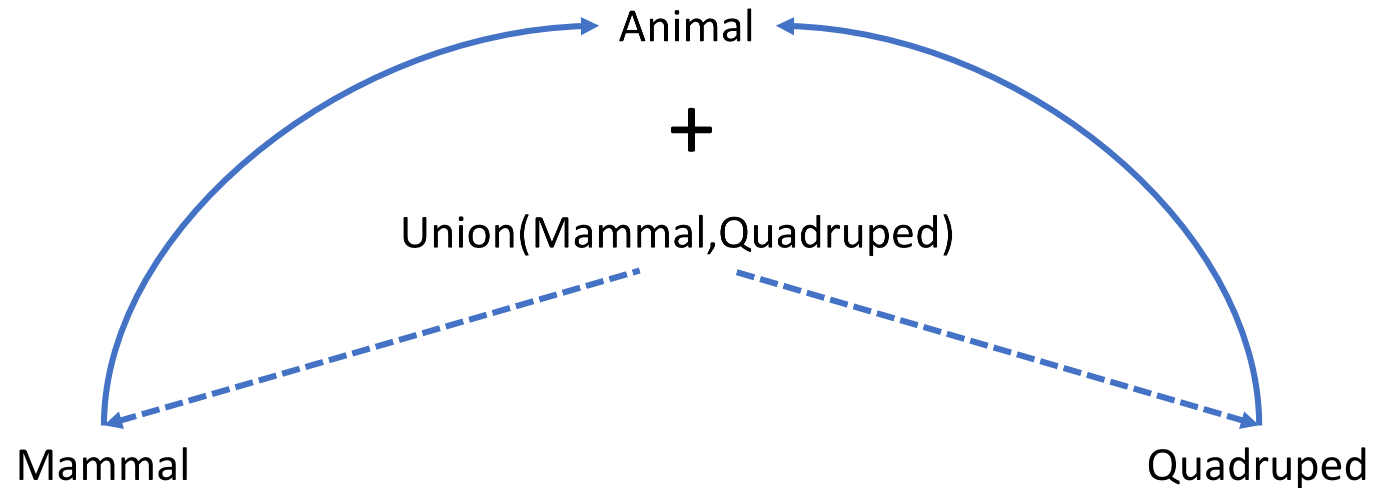
Greatest Lower Bound (glb / meet / inf)

- Intersection in wombat



Up from a Union

- Go down to components then up.
  - The fact that one of the down links must succeed is a defining property of Union.
- Combine results (case procedure).
- Union has the properties of Animal, but not other properties of Mammal or Quadruped.



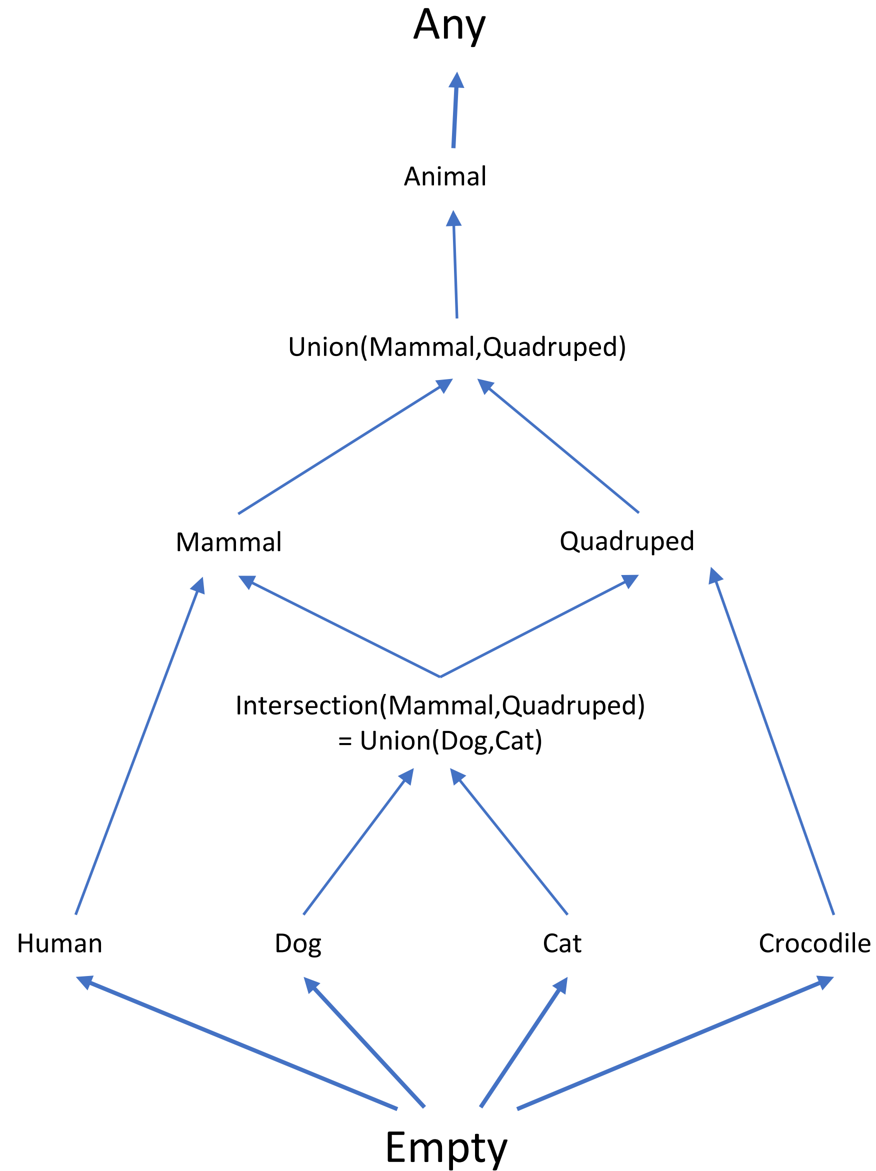
Bounded Lattice

All sets have a lub, including empty set:

- `Union() = Empty`

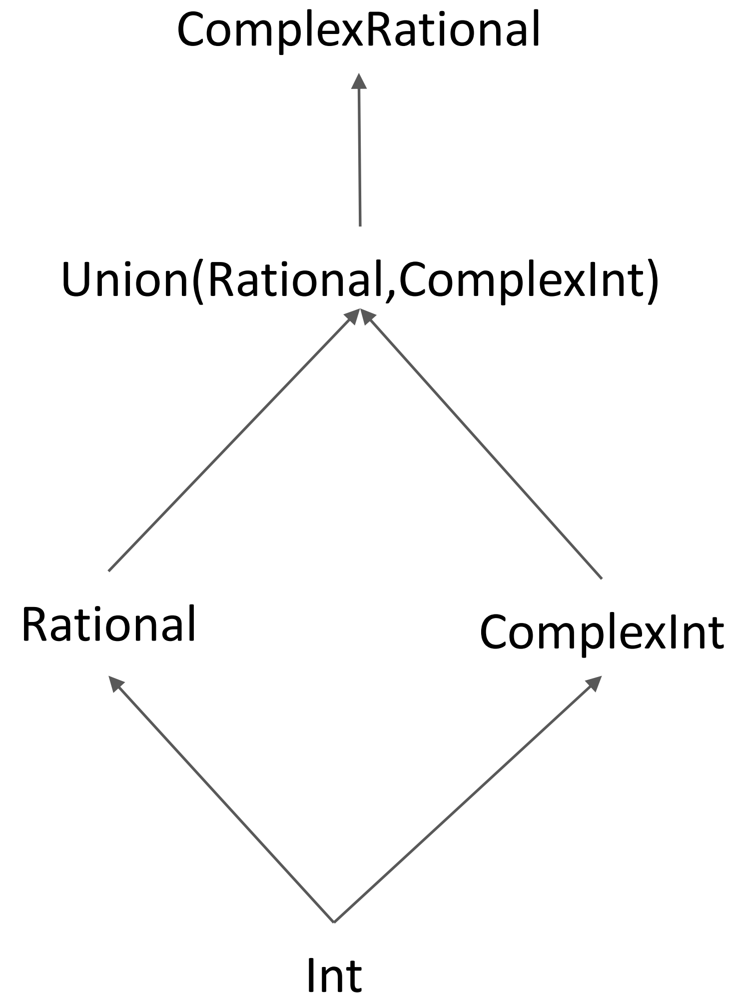
All sets have glb

- `Intersection() = Any`



Properties

- add property must be compatible
- Union inherits add from ComplexRational
  - $ComplexRational^2$=>ComplexRational
- Properties always come in a bundle (Behaviour), but only the separate properties are inherited downwards.
- A property is identified by:
  - Name
  - Behaviour that includes the name
  - Type that conformsTo Behaviour
  - ‘how’ index (because a type can conform in more than one way).



Subset Types

- aka refinement types
- No properties of their own, all properties inherited from the parent type.
- Subset restricted to a single value is an important special case.
  - The only subset type supported in the current implementation.
- If a subtype in the **isA** hierarchy has no additional properties then it is indistinguishable from a subset type.

Example program, language features

```
`test = { case $:Nat of [
               { $ = 0; 100}
               { $ = `n; n-1 }
             ]
           };
 print (test 4);
 6 = test `x; print x;
 100 = test `y; print y
# Backquote when name used for first time
# Explicit closure in braces, {}. $ is the input parameter
# Case takes list of procedures, expects one to succeed.
# = unifies, can fail. n-1 will fail if n:Nat=0.
```

Procedures in the Hierarchy

If

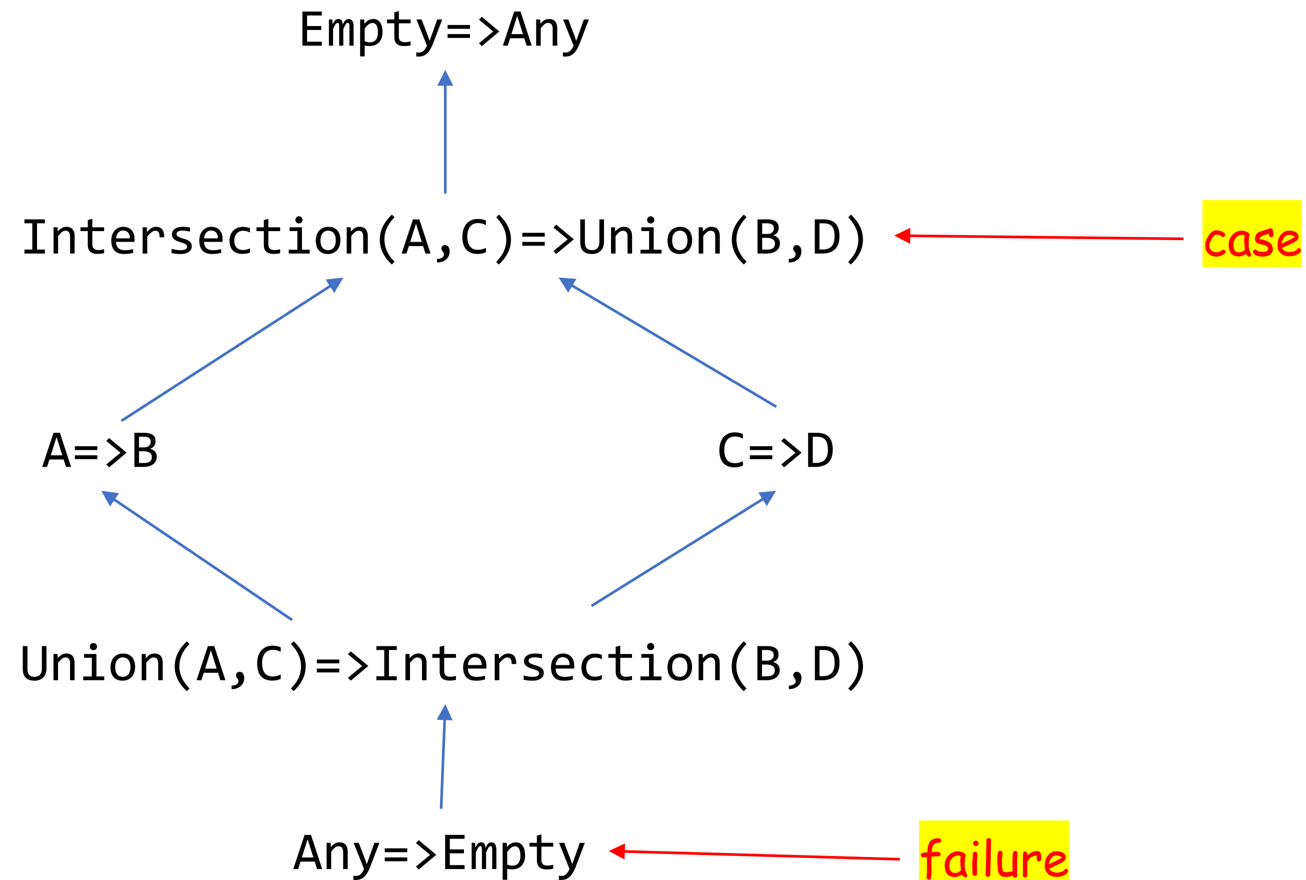
A=>B **isA** X=>Y

Then an `ab::A=>B` can be used where an X=>Y is expected.
So it must accept an X and return a Y. So we must have:

- X **isA** A, and
- B **isA** Y

The conditions are also sufficient.

Procedures form a sub bounded lattice



Bidirectional Type Driven Execution

- All subexpressions in a closure are executed.
- All optional/repeated/async execution of code is by passing closures to an appropriate procedure, such as `whileP`, `ifP`, or `caseP`.
- All subexpressions in the closure execute simultaneously.
  - They start with type Any.
  - As they learn their type is lower they let neighbours know.
  - Neighbours are parent, children, and for local identifiers the id registry.
- Since all changes are monotonically downward this process must terminate.

A simple example

Consider:

```
`x::Nat ; x = 3; x-1
```

- 3 has type Decimal (numbers with a finite number of decimal places – the type of numeric constants) restricted to value 3.
- The = operator sets both sides to the Intersection.
- Then, since the left of the = is an identifier, it reports to the registry, which then reports to all other cases of that identifier.
- Order of operation doesn't matter: all operations commutative.
  - If = before `::` then x set to Decimal/3 before `::` op converts it to Nat/3
  - If `::` before = then = op sees types Nat and Decimal/3: intersection is Nat/3
- Interpreter code (approximate):

```
class PvRequal(PVrun):
    def pTrT(self,pt,rt):
        newt = H.intersectionList([pt.tMindx[0],pt.tMindx[1],rt])
        return MtVal(T.mfTuple,(newt,newt),None),newt
```

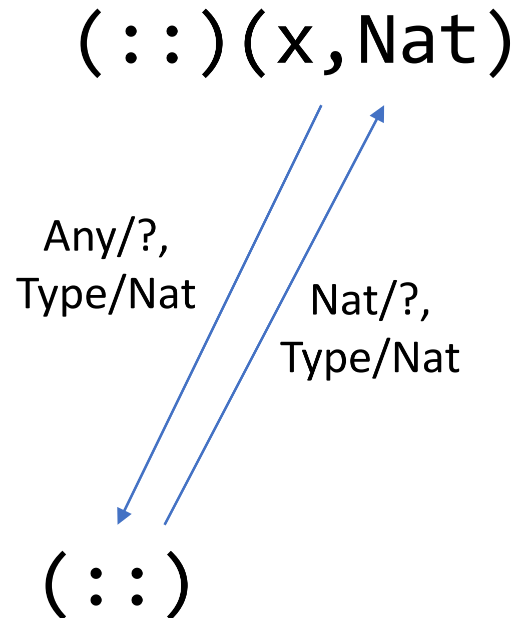
Procedure execution(1)

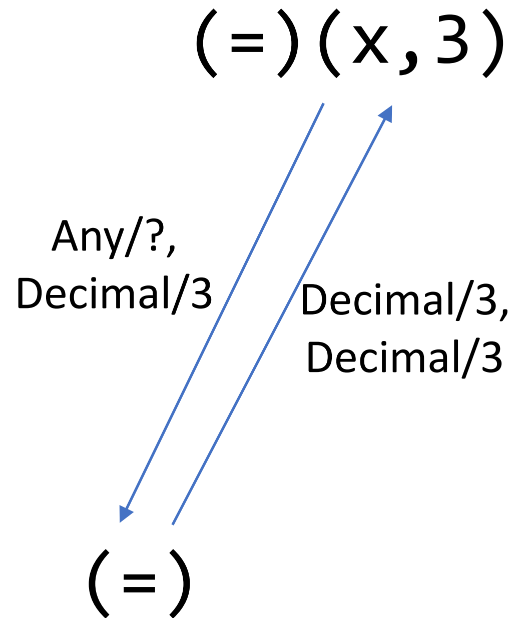
```
{ `x::Nat ; x = 3 ; x-1 }
```

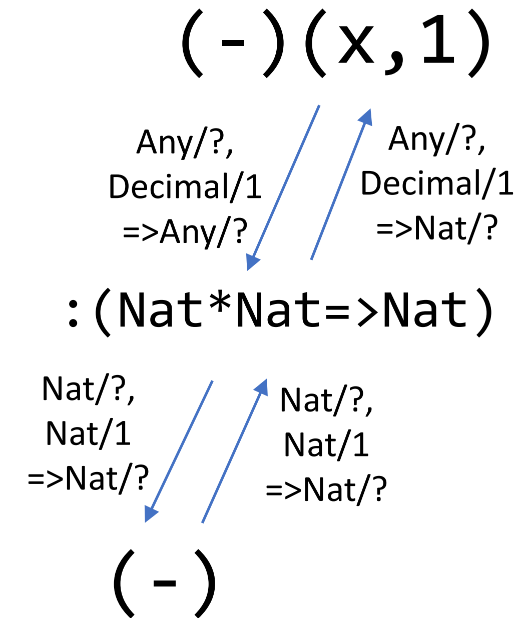
Asssume leftmost will be done first

registry
x :: Any/?

conversion operations don't affect the caller's parameter







various simplifications, e.g. (;) is also a procedure

Procedure execution (2)

```
{ `x::Nat ; x = 3 ; x-1 }
```

only middle can move next

registry

x :: Nat/?

```
(::)(x,Nat)        (=)(x,3)        (-)(x,1)
```

Nat/?,
Type/Nat

Nat/?,
Type/Nat

Nat/?,
Decimal/3

Nat/3,
Nat/3

Nat/?,
Nat/1
=>Nat/?

Nat/?,
Nat/1
=>Nat/?

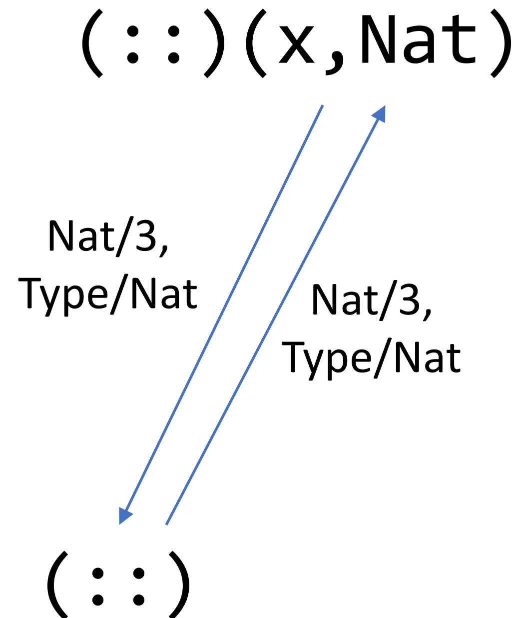
```
(::)        (=)        (-)
```

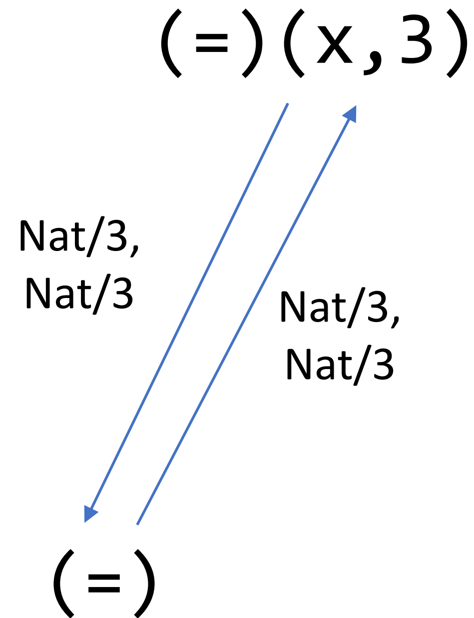
Procedure execution (3)

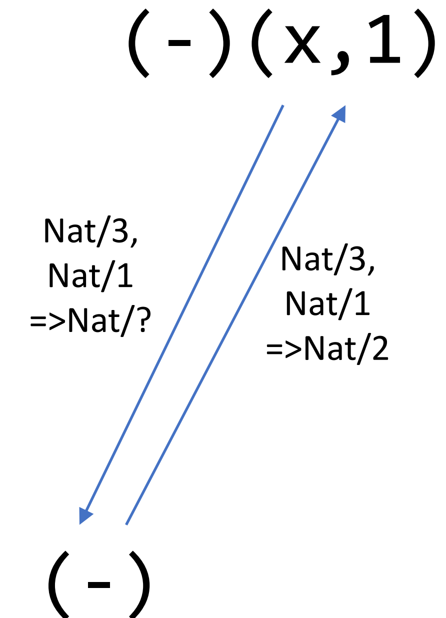
```
{ `x::Nat ; x = 3 ; x-1 }
```

registry

x :: Nat/3







The case operator/procedure

```
case x:Nat of [ { $ = 0; 100}
                { $-1 } ]
```

- Takes a list of procedures and passes input and output to each.
- Normally one succeeds giving the result, others fail.
- When a procedure fails it sets output to Empty, input to Any.
- Case does Intersection of inputs, Union of outputs.
- Allows case statement to run backwards.
- last line of interpreter code for `case`:

```
return L.bind(pt).tMindx[0].set(H.intersectionList(pts)),H.unionList(rts)
```

questions?

Wombat welcomes:

- collaborators
- academic involvement

To come:

- Effect system (stuff that needs to be done in the right order)
- Proof system (minimize run time checks)
- full language + Wombat library
- Documentation

Properties

- The case statement doesn't actually take a List of procedures because the order doesn't matter.
- It doesn't take a Set (or multiset/bag) of procedures because procedures don't fully support equality (f==g can return .unknown).

It is, in wombat-speak, a Semiset. Functions using a Semiset must give the same answer when member values occur more than once. This means that it's defining property must take a commutative idempotent action procedure, and a value for the empty Semiset, and return the fold. Case is consistent with this.

Action: f:R*X=>R, commutative means f(f(r,x1),x2) == f(f(r,x2),x1),
idempotent means f(f(r,x),x) == f(r,x)