NELUMBO

LOGIC META LANGUAGE

- Fast logic Reasoning
- Define and parse Syntaxes
- Define and execute Semantics
- Define and run Tests
- IDE integration using LSP
- Lightweight Language Workbench

REQUIREMENTS FOR NELUMBO

- Suitable for formalization of, for example, complex tax laws and clinical knowledge
- Semantically rich and proven consistent
- Direct support for predicate logic by reasoning over incomplete facts and falsehoods
- Fully declarative, hence, no over-specification
- Strongly typed for more consistency and extensibility
- Performant by binding variables based on navigating relations only
- Easily extensible and integrable with the use of native classes
- Full editor support in any IDE using the Language Server Protocol

SYNTAX

TYPES

```
<Smart>    :: <Node>
<Living>   :: <Node>
<LLM>      :: <Smart>
<Person>   :: <Smart>, <Living>
<Male>     :: <Person>
<Female>   :: <Person>
```

PATTERNS

```
<Integer> ::= <NUMBER>                              // 10
<Integer> ::= <Integer> - <Integer>  #40,           // 5-7
              <Integer> + <Integer>  #40,           // 5+7
                        - <Integer>  #80,           // -7
              fib(<Integer>)                        // fib(100)

<Repetion>     ::= { <(> <Integer> <,> , <)*> }  // {3,5,7}
<Option>       ::= <(> super <)?> fast           // fast, super fast
<Alternation>  ::= <(> A <|> B <|> C <)>         // A, B, C
```

SEMANTICS

VARIABLES

```
<Integer> a, b, c
<Person>  x, y, z
```

RULES

```
a<=b     <=>  a<b | a=b

|a|=b    <=>  a>=0 & b=a |
              a<0 & b=-a

descendant(x)=z <=> child(x)=z |
                    E[y](descendant(x)=y & child(y)=z)
```

RUNNING

QUERIES

```
a+11=21   ?     // [(a=10)][..]
```

TESTING

```
a+11=21   ?   [(a=10)][..]
|a|=10    ?   [(a=-10),(a=10)][..]
```

PROOF

```
a+11=21
  a+11=21 <=> add(a,11,21)
    add(c,11,21) [(c=10)][..]
  c+11=21 [(c=10)][..]
a+11=21 [(a=10)][..]
```

NATIVE SEMANTICS

DECLARATION

```
<Predicate> ::= add(<Integer>,<Integer>,<Integer>)
                @org.modelingvalue.nelumbo.integers.Add
```

JAVA CODE

```
protected InferResult infer(int nrOfUnbound, InferContext context) {
   if (nrOfUnbound > 1) {
       return unknown();
   }
   BigInteger addend1 = getVal(0, 0);
   BigInteger addend2 = getVal(1, 0);
   BigInteger sum = getVal(2, 0);
   if (addend1 != null && addend2 != null) {
       BigInteger s = addend1.add(addend2);
       if (sum != null) {
           boolean eq = s.equals(sum);
           return eq ? factCC() : falsehoodCC();
```

FIBONACCI EXAMPLE

```
<Integer> ::= fib(<Integer>)

<Integer> n, f

fib(n)=f <=> f=n                     if n<=1,
             f=fib(n-1)+fib(n-2) if n>1

fib(0)=f       ? [(f=0)][..]
fib(1)=f       ? [(f=1)][..]
fib(5)=f       ? [(f=5)][..]
fib(100)=f     ? [(f=36#22r8fozas3n8w3)][..]
fib(1000)=f    ? [(f=36#18nrvsuavughau0blk8avlvbvaqwiaqha77rdsgscn5hzwghgaws8
```

LITERALS

```
<Male>    ::= Hendrik, Bernhard, Claus, Willem
<Female>  ::= Wilhelmina, Juliana, Beatrix, Maxima, Amalia

<Integer> ::= <NUMBER>    @org.modelingvalue.nelumbo.integers.Integer
```

FUNCTIONS

```
<Integer> ::= <Integer> -  <Integer>  #40,
              <Integer> +  <Integer>  #40

a+b=c  <=>  add(a,b,c)
a-b=c  <=>  add(c,b,a)
```

RELATIONS AND FACTS

```
<Relation>  ::= pc(<Person>,<Person>)   // parent-child

// Facts
pc(Beatrix, Willem)
pc(Claus, Willem)
pc(Willem, Amalia)
pc(Maxima, Amalia)
```

PREDICATES

```
<Predicate> ::= <Integer> <= <Integer>  #30

a<=b    <=>  a<b | a=b
```

Operator	Meaning
`::`	Type
`::=`	Pattern
`<=>`	Rule

DEMO

LIVE

PLANS

- Namespaces
- Generics (type arguments)
- Lists and Sets
- Language pattern transformations
- LSP (also on WEB)
- Reactive update execution semantics
- Deprecation and migration support
- ....

CONTRIBUTING

- Open source: create tests and libraries
- Try to falsify the logic
- Help write scientific publications
- GitHub: https://github.com/ModelingValueGroup/nelumbo
- Email: wim.bast@gmail.com

Q&A