Meeting 10 - Judgments

Bor-Yuh Evan Chang

Thursday, September 26, 2024

Meeting 10 - Judgments

WAP WAP WAP
CALVIN! WHAT ARE YOU DOING TO THE COFFEE TABLE ?!?
IS THIS SOME SORT OF TRICK QUESTION, OR WHAT?
WATTERSON

What questions does your neighbor have?

- In-Class Slides
- In-Class Jupyter
- Book Chapter

Announcements

- Lab 2 due next Monday (9/30) 6pm

Today

- Judgments
- Triage Your Questions

Questions?

Review: Type Judgments

$$e : \tau$$

Inference rules

- We define judgment forms inductively using a set of *inference rules*:

$$\frac{J_1 \qquad J_2 \qquad \cdots \qquad J_n}{J}$$

- If we can show $J_1, J_2, \ldots J_i$, then J holds, i.e. we can *derive* J - the conclusion

Example - Syntax of natural numbers

$$\boxed{n \in \mathbf{Nat}}$$

$$\text{Zero}$$

$$\frac{}{\textsf{z} \in \mathbf{Nat}}$$

$$\text{Successor}$$

$$\frac{n \in \mathbf{Nat}}{\textsf{s}(n) \in \mathbf{Nat}}$$

Let’s code it

```scala
sealed abstract class Nat
```

```
defined class Nat
```

```scala
def isNat(n: Nat): Boolean = ???
```

```
defined function isNat
```

- This is the characteristic function of the set $\mathbf{Nat}$
- Notice the correspondence between the mathematical specification and the implementation

Derivations

- A judgment holds if and only if we can compose applications of the inference rules to demonstrate it

$$\mathsf{s}(\mathsf{s}(\mathsf{z}))$$

- this can be represented in code, as each judgment corresponds to a function call

```
def isNatDerivation(n: Nat): Boolean = ???
```

```
defined function isNatDerivation
```

Structural Equality

$$\boxed{n_1 =_{\mathbf{Nat}} n_2}$$

```scala
def eqNat(n1: Nat, n2: Nat): Boolean = ???
```

```
defined function eqNat
```

Semantics

- What if we want to turn our syntactical evaluation of $\mathbf{Nat}$s into a semantical evaluation by deriving what integer value a $\mathbf{Nat}$ holds?
- What is our judgment form?
- What does $\mathbf{z}$ evaluate to?
- For our successor inference rule: given ??? as a premise, what does ??? evaluate to?

```
def eval(n: Nat): Int = ???
```

```
defined function eval
```

Let’s try lists

- Inductively define with judgments and inference rules

```
sealed abstract class MyIntList
```

```
defined class MyIntList
```

```
def isMyIntList (l: MyIntList) : Boolean = ???
```

```
defined function isMyIntList
```

List equality

```scala
def eqMyIntList (l1 : MyIntList, l2: MyIntList) : Boolean = ???
```

```
defined function eqMyIntList
```