

Advanced Topics with Programming Languages

Josh Felmeden

September 28, 2021

Contents

1	Introduction	3
2	Judgements	3
2.1	Simultaneous rules	3
3	Induction	4
3.1	Simultaneous induction	4

1 Introduction

One way of looking at programming languages is to look at **types** and **type systems**. Haskell is a language that uses typing. There can be static and dynamic typing.

Types classify programs by the kind of data they compute.

2 Judgements

A **judgement** is a statement. In this topic, we will centre everything around an *evident judgement*. A judgement becomes evident when you can *prove* it. Therefore, when we say a judgement, we need to provide evidence of proof.

Judgements come with rules. Here is an axiom:

```
zero nat
```

Zero is the object, and nat is the name. Alongside this, we can use an inference rule:

```
n nat           'premise(s)
----- s 'name of rule
succ(n) nat      'conclusion
```

These two structures can be used in **derivation trees** which are used to prove judgements. For example, to prove that two is a natural number, we can do the following:

```
----- axiom
zero nat
----- s
succ(zero) nat
----- s
succ(succ(zero)) nat
```

We can also write

```
data nat = zero | succ nat
```

2.1 Simultaneous rules

we can state proofs of rules mix and match to use a proof that proves two things at once. For example:

```
----- ZE
zero even
```

```
n even
----- ODD
succ(n) odd
```

```
n odd
----- EVEN
succ(n) even
```

This proves both odd and even.

3 Induction

Every set of rules generates an *induction principle*.

Consider the claim **if** `succ(n)nat` **then** `n nat`. This seems obvious, but we can actually prove this.

Proof We will use induction

$P(n)$: 'If `n nat` and `n = succ(x)` for some `x` then `x nat`'

Case zero: Nothing to prove

Case (`succ(n) nat`) The derivation of `succ(n) nat` ends with

```
n nat
----- succ
succ(n) nat
```

The D is a derivation of `n nat`.

`succ(n) = succ(x)` and therefore `n = x`. We can conclude that `n` is `nat` and therefore `x` is `nat`.

This statement is an **admissible rule**. A rule is admissible when we have a derivation of the premises, then we know we can construct a derivation of the conclusion. In essence, you need to *prove* this one (usually by induction).

In contrast, a rule is **derivable** if we can use a derivation of its premise as a building block in deriving its conclusion. In essence, you can *infer* this one (stitch together stuff).

3.1 Simultaneous induction

Recalling the even and odd proof, we can write these as Let $P(n \text{ even})$ and $Q(n \text{ odd})$. If:

- $P(\text{zero})$ and
- whenever n even and $P(n)$ we have $Q(\text{succ}(n))$ and
- whenever n odd and $Q(n)$ we have $P(\text{succ}(n))$

We are allowed to *invert* a judgement, and this is called an *inversion principle*.