Inductive Definition

OUTLINE

- Judgements
- Inference Rules
- Inductive Definition
- Derivation
- Rule Induction

Language and Meta-language

- Language is the target programming language, e.g., Java, Python, ML.
  - Has its own identifiers, variables, etc.

- Meta-language is the language in which to describe the target language.

Meta-Variables

- A symbol in a meta-language that is used to describe some element in an object (target) language
  - E.g., Let **a** and **b** be two sentences of a language $\mathcal{L}$
  - E.g., Let **n** be a number, **d** be a digit and **s** be a sign in the language of numerals
    - 435, 535.23, -3847 are all numbers in the language of numerals
  - meta-variable doesn't appear in the language itself.
- Meta- is a prefix used to indicate a concept, which is an abstraction from another concept, used to complete or add to the latter.
- Similar use in "meta-data", "meta-theory", etc.
  - The syntax, semantics, etc. about a PL (e.g., Java) is the meta-theory about that language

Judgements

- A *judgement* is an *assertion* (in the meta-language) about one or more syntactic objects.

Judgement	Meaning
n nat	(n is a natural number)
$n = n_1 + n_2$	(n is the sum of n_1 and n_2)
τ type	(τ is a type)
$e:\tau$	(expression e has type τ)
$e \Downarrow v$	(expression e has value v)

- "n nat" can also be written as "n isa nat", "n is a natural num", etc. as long as it's consistent.

JUDGEMENTS (II)

- A judgement states one or more syntactic objects have a property or have a relation among one another.
- The property or the relation itself is called *predicate*.
  - E.g., n nat (this judgement involves one object n)
- The abstract structure (schema) of a judgement is called *judgement form*.
  - E.g. n nat.
- The judgement that a particular object or objects having that property is an *instance* of a judgement form.
  - E.g., 5 nat, succ(n) nat are all judgements
- W.L.O.G., we use "judgement" to mean the instance of judgement form usually.

Inference Rules

- An inductive definition of a judgement form consists of a collection of rules of the form:

$$\frac{J_1 \quad \ldots \quad J_k}{J}$$

premises

conclusion

- To show J, it is sufficient to show J_1, …, J_k.
- A rule without premises is called an *axiom*;
- Otherwise it's called a *proper rule*.

Inductive Definition

- Definition of judgement form *n* nat:

$$\frac{}{zero \quad nat} \qquad \frac{n \quad nat}{succ(n) \quad nat}$$

- Definition of judgement form *t* tree:

Axioms!

Proper Rules!

$$\frac{}{empty\ tree} \qquad \frac{t_1\ tree \quad t_2\ tree}{node(t_1;\ t_2)\ \ tree}$$

Derivation

- To show an inductively defined judgement holds → exhibit a derivation of the judgement.
- A derivation is an *evidence* for the validity of the defined judgement.
- Derivation of a judgement is the finite composition of rules starting from *axioms* and ending at *that judgement*.
- Usually a tree structure
  - In compiler, derivation of grammar in the form of a *parse tree*.

Derivation (II)

- Derivation of judgement succ(succ(succ(zero))) nat:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{zero\ \ nat}}{succ(zero)\quad nat}}{succ(succ(zero))\quad nat}}{succ(succ(succ(zero)))\quad nat}$$

- Derivation of node(node(empty, empty), empty) tree:

$$\dfrac{\dfrac{\dfrac{}{empty\quad tree}\qquad \dfrac{}{empty\quad tree}}{node(empty; empty)\quad tree}\qquad \dfrac{}{empty\quad tree}}{node(node(empty; empty); empty)\quad tree}$$

Types of Derivation

- Forward chaining (bottom-up):
  - Starting from axioms, work up to the conclusion
- Backward chaining (top-down):
  - Start from the conclusion, work backwards toward axioms
- Note the terms bottom-up and top-down are exactly the opposite of the derivation tree we presented.

Type of Derivation

- Derivation of judgement succc(succ(succ(zero))) nat:

Backward chaining

$$\dfrac{\dfrac{\dfrac{zero\ \ nat}{succ(zero)\quad nat}}{succ(succ(zero))\quad nat}}{succ(succ(succ(zero)))\quad nat}$$

Forward chaining

Deductive Systems

- A deductive system has 2 parts:
  - Definition of a judgement form or a collection of judgement forms
  - A collection of inference rules about these judgement forms

- We have just introduced two deductive systems: nat and tree.

- A *programming language* can be represented by a deductive system, of course with many judgement forms!

Rule Induction (I)

- Reason about rules under an inductive definition (or within a deductive system)
- Principle of rule induction:
  - To show property P holds of a judgement form J whenever J is derivable, it is enough to show that P *is closed under*, or *respects*, the rules defining J.
  - Write P(J) to mean property P holds for J.
  - P respects the rule

$$\frac{J_1 \quad \ldots \quad J_k}{J}$$

  if P(J) holds whenever $P(J_1)$, ..., $P(J_k)$ hold.
  - $P(J_1)$, ... $P(J_k)$ are *inductive hypothesis*.
  - P(J) is *inductive conclusion*.

Rule Induction (II)

- For the judgement n nat, to show P(n nat), it is sufficient to show:
    1. P(zero nat).
    2. For every n, if P(n nat), then P(succ(n) nat).
- Looks familiar?
- This is just a generalized version of *mathematical induction*.
- Step 1 is called the basis; step 2 is called the induction step.
- Similar induction can be applied on node(t_1, t_2) tree → "*tree induction*".

Proof By Induction

Outline

- Proof Principles
- Natural Numbers
- List
- Proof Structure

Proof Principle (Rule Induction)

- Recall that…
- To show every derivable judgement has some property P, show for every rule in the deductive system:

$$\frac{J_1 \ldots J_n}{J}[name]$$

- If J_1, …, J_n have property P then J has property P.

Example (Natural Numbers)

- Given a property P, we know that P is true for all natural numbers, if we can prove:
  - P holds unconditionally for Z. Corresponds to rule Z:

$$\frac{}{Z \quad nat}\,Z$$

  - Assuming P holds for n, then P holds for (S n). Corresponds to rule S:

$$\frac{n \quad nat}{S\, n \quad nat}\,S$$

- Also called “induction on the structure of natural numbers”.

Natural Numbers

- Natural numbers:

$$\frac{}{Z \quad nat}\,Z \qquad\qquad \frac{n \quad nat}{S\,n \quad nat}\,S$$

- Addition:
  - Judgement: add n1 n2 n3

$$\frac{}{add\ Z\ n\ n}\,addZ \qquad\qquad \frac{add\ n1\ n2\ n3}{add\ (S\ n1)\ n2\ (S\ n3)}\,addS$$

Theorem 1: For all n1, n2, there exists n3 such that add n1 n2 n3.
(*if n1 nat, n2 nat, then there exists n3 nat such that add n1 n2 n3*)
Proof: **By induction on the derivation of n nat.**

Case: $$\frac{}{Z \quad nat}\, Z$$

Need to prove add n1 n2 n3 where n1 = Z

(1) add Z n2 n2 (by addZ, and let n=n2)

(2) add n1 n2 n3 (by letting n1=Z, n3=n2)

Renaming!

(Case proved)

Case: $$\frac{n \quad nat}{S\, n \quad nat}\, S$$

$$\frac{}{add\ Z\ n\ n}\, addZ$$

$$\frac{add\ n1\ n2\ n3}{add\ (S\ n1)\ n2\ (S\ n3)}\, addS$$

Need to prove add n1 n2 n3 where n1 = (S n)

(1) add n n2 n3’ (by I.H. and let n = n1, n3’=n3)

(2) add (S n) n2 (S n3’) (by (1), addS, and
let (S n) = n1, (S n3’)=n3)

(Case proved) QED.

Even/Odd Numbers

- Judgements:
  - even n “n is an even number”
  - odd n “n is an odd number”

$$\frac{}{even \quad Z}\, evenZ \qquad \frac{odd \quad n}{even \quad (S\, n)}\, evenS$$

$$\frac{even \quad n}{odd \quad (S\, n)}\, oddS$$

Theorem 2: If n nat, then either even n or odd n.

$$\frac{}{even \quad Z}\, evenZ$$

$$\frac{odd \quad n}{even \quad (S\, n)}\, evenS$$

$$\frac{even \quad n}{odd \quad (S\, n)}\, oddS$$

Proof: **By induction on the derivation of n nat.**

Case: $$\frac{}{Z \quad nat}\, Z$$

even Z (By rule evenZ)

Case: $$\frac{n \quad nat}{S\, n \quad nat}\, S$$

(1) even n or (2) odd n (By I.H.)

Need to prove: even (S n) or odd (S n)

Assuming (1):

odd (S n) (By (1) and rule oddS)

Assuming (2):

even (S n) (By (2) and rule evenS)

QED.

Even/Odd Number (Alt. Definition)

$$\frac{}{even2 \quad Z}\ even2Z \qquad \frac{even2 \quad n}{even2 \quad (S\ (S\ n))}\ even2S$$

$$\frac{}{odd2 \quad (S\ Z)}\ odd2Z \qquad \frac{odd2 \quad n}{odd2 \quad (S\ (S\ n))}\ odd2S$$

$$\frac{}{even \quad Z}\, evenZ$$

$$\frac{odd \quad n}{even \quad (S\,n)}\, evenS$$

$$\frac{even \quad n}{odd \quad (S\,n)}\, oddS$$

Theorem 3: If even2 n, then even n.

Proof: By induction on the derivation of even2 n.

Case: $$\frac{}{even2 \quad Z}\, even2Z$$

even Z (by rule evenZ)

Case: $$\frac{even2 \quad n}{even2 \quad (S\,(S\,n))}\, even2S$$

(1) even n (by I.H.)

Need to prove: even (S (S n))

(2) odd (S n) (by (1), oddS)

(3) even (S (S n)) (by (2), evenS)

QED.

List of Natural Numbers

- Judgement Form:
  - l list "l is a list"

$$\frac{}{nil \quad list}\, nil \qquad\qquad \frac{n\ nat \quad l\ list}{cons(n,l) \quad list}\, cons$$

- Cons stands for "CONcatenateS"
- Means concatenation of a *head* and a *tail* of a list.
- In cons(n, l), n is the head and l is the tail.
- cons(1, cons(2, cons(3, nil))) = 1::2::3::nil = [1,2,3]

Lemma 1: cons((S Z), cons(Z, nil)) is a list.

Proof: By giving a derivation of cons((S Z), cons(Z, nil)) list.

```
  Z nat   (by Z)                     Z nat  (by Z)     nil list  (by nil)
--------------------- (by S)    ---------------------------------------------   (by cons)
    (S Z) nat                                cons(Z, nil) list
---------------------------------------------------------------------------  (by cons)
                 cons((S Z), cons(Z, nil))  list
```

List - Len

- Judgment Form: len l n.
  - "the length of l is n".

$$\frac{}{len\ nil\ Z}\ len-nil$$

$$\frac{len\ l\ n}{len\ cons(n_1,l)\ (S\ n)}\ len-cons$$

List - Append

- Judgment Form: append l_1 n l_2.
  - "l_2 is the result of appending n to l_1".

$$\frac{}{append\ nil\ \ n\ \ cons(n, nil)}\ append - nil$$

$$\frac{append\ l\ n_2\ l_1}{append\ cons(n_1,\ l)\ \ n_2\ \ cons(n_1,\ l_1)}\ append - cons$$