COMPSCI/SFWRENG 2FA3
Discrete Mathematics with Applications II
Winter 2018

1 Recursion and Induction

William M. Farmer

Department of Computing and Software
McMaster University

February 7, 2018

Admin — January 8

- Tutorials and M&Ms start this week.
- Continuity with CS/SE 2DM3.
  - Same notation when possible.
  - Some exercises and assignments will use CalcCheck.
  - Some topics will be re-examined from a more abstract vantage point.
- We will start by re-examining recursion and induction.
- Office hours: To see me please send me a note with times.
- Are there any questions?

What is Recursion?

- Recursion is a method of defining a function or structure in terms of itself.
  - One of the most fundamental ideas of computing.
  - Can make specifications, descriptions, and programs easier to express, understand, and prove correct.
- A problem is solved by recursion as follows:
  1. The simplest instances of the problem are solved directly.
  2. Each other instance of the problem is solved by reducing the instance to simpler instances of the problem.
  3. As a result of 1 and 2, each instance can be solved by reducing the instance to simpler instances and then reducing these instances to simpler instances and continuing in this fashion until a simplest instance is reached, which has already been solved.
- Recursion employs a divide and conquer strategy.

How does Recursion Work with Functions?

- In the typical recursive definition of a function:
  - An instance of the function is a tuple of inputs for the function.
  - Each instance I is assigned a natural number $n(I)$.
  - An instance I is a "simplest instance" if $n(I) = 0$.
  - An instance I' is "simpler than an instance I if $n(I') < n(I)$.
- A recursive definition of a function is nonsensical if some instance I is reduced to an instance I' such that I' is not simpler than I, i.e., $n(I') \geq n(I)$.

What is Induction?

- Induction is a method of proof based on a recursively defined structure or a well-founded relation.
  - Most important proof technique used in computing.
  - The proof method is specified by an induction principle.
  - Induction is especially useful for proving properties about recursively defined functions.
- Note: The terms "recursion" and "induction" are often used interchangeably.

Two Styles of Recursion and Induction

- Structural recursion and induction
  - Based on an inductive type.
  - Statements are proved by a structural induction principle.
  - Functions can be defined by pattern matching.
- Well-founded recursion and induction
  - Based on a well-founded relation.
  - Statements are proved by a well-founded induction principle.
  - Functions can be defined by well-founded recursion.

Structural Recursion and Induction [1/2]

- An inductive type is a type t defined by a finite set of constructors (where $m_1, \ldots, m_n \geq 0$)

$$\begin{array}{l} C_1 : t^1_1 \times \cdots \times t^1_{m_1} \to t. \\ \quad \vdots \\ C_n : t^n_1 \times \cdots \times t^n_{m_n} \to t. \end{array}$$

  such that each value of a of type t can be constructed from the constructors in exactly one way.
  - That is, "no junk and no confusion".
- Some of the types $t^1_1, \ldots, t^n_{m_n}$ may be t itself.
  - In this case, t is said to be recursive.
- The constructors $C_1, \ldots, C_n$ define a language whose expressions serve as literals for the members of t.

Structure Recursion and Induction [2/2]

- The definition of t induces a structural induction principle: A property P holds for all members of t provided for every constructor C_i

  if P holds for every x_j of type t in $C_i(x_1, \ldots, x_{m_i})$, then P holds for $C_i(x_1, \ldots, x_{m_i})$.

  Less formally, a property P holds for all members of t provided:

  1. P holds for all members of S having minimal structure.
  2. P holds for a structural combination of members of t whenever it holds for the members themselves.

- A function f on t can be defined by pattern matching.
  - Each recursive application of f must be applied to at least one argument with reduced structure.

Natural Numbers (iClicker)

How many constructors are needed to define the natural numbers as an inductive type?

A. 1.

B. $\boxed{2.}$

C. 3.

D. 4.

Example 1: Natural Numbers as an Inductive Type

- Nat is the inductive type representing the natural numbers defined by the following constructors:
  1. $0 : \mathsf{Nat}$ (i.e., $0 : \to \mathsf{Nat}$).
  2. $S : \mathsf{Nat} \to \mathsf{Nat}$.
- Nat is recursive.
- The members of Nat correspond to the expressions

  $$0, S\,0, S\,(S\,0), \ldots$$

  which denote the natural numbers

  $$0, 1, 2, \ldots.$$
- The structural induction principle for Nat is:

  $$(P\,0 \wedge (\forall\, x : \mathsf{Nat} \bullet P\,x \Rightarrow P\,(S\,x))) \Rightarrow (\forall\, x : \mathsf{Nat} \bullet P\,x)$$

  holds for every property P of Nat. This principle is called mathematical induction or weak induction.

Example 1: Functions Defined by Pattern Matching

- Addition ($+ : \mathsf{Nat} \times \mathsf{Nat} \rightarrow \mathsf{Nat}$) is defined by pattern matching as:
  1. $x + 0 = x$.
  2. $x + S\,y = S\,(x + y)$.
- Multiplication ($* : \mathsf{Nat} \times \mathsf{Nat} \rightarrow \mathsf{Nat}$) is defined by pattern matching as:
  1. $x * 0 = 0$.
  2. $x * S\,y = (x * y) + x$.
- The function $\mathsf{fib} : \mathsf{Nat} \rightarrow \mathsf{Nat}$ that maps n to the nth Fibonacci number is defined by pattern matching as:
  1. $\mathsf{fib}\,0 = 0$.
  2. $\mathsf{fib}\,S\,0 = S\,0$.
  3. $\mathsf{fib}\,S\,(S\,x) = \mathsf{fib}\,(S\,x) + \mathsf{fib}\,x$.

Example 1: Proof of $\forall\, x : \mathrm{Nat} \bullet 0 + x = x$

1. Let $P\, x \equiv 0 + x = x$.
2. Base case: Show $P\, 0$.
   - 2.1 $P\, 0 \equiv 0 + 0 = 0$ by the definition of P.
   - 2.2 $0 + 0 = 0$ is an instance of $x + 0 = x$.
   - 2.3 Hence $P\, 0$ holds.
3. Induction step: Assume $P\, x$ holds. Show $P\, (S\, x)$.
   - 3.1 $P\, (S\, x) \equiv 0 + S\, x = S\, x$ by the definition of P.
   - 3.2 $0 + S\, x = S(0 + x)$ is an instance of $x + S\, y = S\, (x + y)$.
   - 3.3 $0 + x = x$ by the induction hypothesis $P\, x$.
   - 3.4 Hence $P\, (S\, x)$ holds.
4. Therefore, $\forall\, x : \mathrm{Nat} \bullet P\, x$ holds by mathematical induction.

Base Cases (iClicker)

A constructor $C : t_1 \times \cdots \times t_m \to t$ in the definition of an inductive definition produces a base case if

A. There are no t_i (i.e., C is 0-ary).

B. $\boxed{\text{None of the } t_i \text{ are } t.}$

C. Some of the t_i are t.

D. All of the t_i are t.

Admin — January 10

- Discussion session on Friday.
- Assignment 1 will be posted at the end of the week.
  - You will submit two files: a LaTeX source file and a PDF output file.
  - How to write documents with LaTeX will be discussed in next week's tutorial. Bring your laptop!
- Office hours: To see me please send me a note with times.
- Are there any questions?

Review — January 10

- Recursion and induction.
- Inductive types.
- Structural induction.
- Natural numbers example.

Example 2: Binary Trees of Natural Numbers

- BinTree is the inductive type representing binary trees of natural numbers defined by the following constructors:
  1. $\mathsf{Leaf} : \mathsf{Nat} \to \mathsf{BinTree}$.
  2. $\mathsf{Branch} : \mathsf{BinTree} \times \mathsf{Nat} \times \mathsf{BinTree} \to \mathsf{BinTree}$.
- BinTree is recursive.
- The structural induction principle for BinTree is:

$$\begin{array}{l}(\forall\, n : \mathsf{Nat} \bullet P\,(\mathsf{Leaf}\, n) \wedge \\ \;(\forall\, n : \mathsf{Nat} \bullet \forall\, t_1 : \mathsf{BinTree} \bullet \forall\, t_2 : \mathsf{BinTree} \bullet \\ \qquad P\, t_1 \wedge P\, t_2 \Rightarrow P\,(\mathsf{Branch}\, t_1, n, t_2))) \\ \quad \Rightarrow (\forall\, t : \mathsf{BinTree} \bullet P\, t)\end{array}$$

  holds for every property P of BinTree.

Example 2: Functions Defined by Pattern Matching

- The function $\mathsf{nodes} : \mathsf{BinTree} \to \mathsf{Nat}$ that maps a binary tree to the number of nodes in it is defined by pattern matching as:
  1. $\mathsf{nodes}\,(\mathsf{Leaf}\ n) = 1$.
  2. $\mathsf{nodes}\,(\mathit{Branch}\ t_1\ n\ t_2) = (\mathsf{nodes}\ t_1) + 1 + (\mathsf{nodes}\ t_2)$.
- The function $\mathsf{sum} : \mathsf{BinTree} \to \mathsf{Nat}$ that maps a binary tree to the sum of the natural numbers attached to its nodes is defined by pattern matching as:
  1. $\mathsf{sum}\,(\mathsf{Leaf}\,\mathsf{n}) = n$.
  2. $\mathsf{sum}\,(\mathit{Branch}\ t_1\ n\ t_2) = (\mathsf{sum}\ t_1) + n + (\mathsf{sum}\ t_2)$.
- The function $\mathsf{height} : \mathsf{BinTree} \to \mathsf{Nat}$ that maps a binary tree to its height is defined by pattern matching as:
  1. $\mathsf{height}\,(\mathsf{Leaf}\,\mathsf{n}) = 0$.
  2. $\mathsf{height}\,(\mathit{Branch}\ t_1\ n\ t_2) = 1 + \max(\mathsf{height}\ t_1, \mathsf{height}\ t_2)$.

Binary Trees (iClicker)

Let t be a member of BinTree. Which of the following formulas is not true?

A. $\boxed{\mathsf{nodes}\, t = 2^{\mathsf{height}\, t}.}$

B. $\boxed{\mathsf{nodes}\, t \le 2^{\mathsf{height}\, t}.}$

C. $\boxed{\mathsf{nodes}\, t = 2^{(\mathsf{height}\, t)+1} - 1.}$

D. $\mathsf{nodes}\, t \le 2^{(\mathsf{height}\, t)+1} - 1.$

Well-Founded Relations

- Let R be a binary relation on U and $S \subseteq U$.
- y is an R-minimal element of S if $y \in S$ and $\forall x \bullet x \in S \Rightarrow \neg(x \; R \; y)$.
- (U, R) is well founded if every nonempty subset of U has an R-minimal element.
  - Examples: $(\mathbb{N}, <)$, $(U \times U, <_{\mathrm{lex}})$.
- A sequence $\langle x_0, x_1, x_2, \ldots \rangle$ of members of U is a descending R-sequence if

  $$\cdots x_2 \; R \; x_1 \; R \; x_0.$$

- (U, R) is noetherian if every descending R-sequence of members of U is finite.
- Theorem. (U, R) is well founded iff (U, R) is noetherian.

Well-Founded Relations (iClicker)

Let H be the set of humans that have lived on earth during the last 100,000 years. Which of the following binary relations on H is well founded?

A. $\boxed{h \; R_1 \; h' \text{ iff } h \text{ is an ancestor of } h'.}$

B. $\boxed{h \; R_2 \; h' \text{ iff } h \text{ is a parent of } h'.}$

C. $\boxed{h \; R_3 \; h' \text{ iff } h \text{ is a child of } h'.}$

D. $\boxed{h \; R_4 \; h' \text{ iff } h \text{ was born before } h'.}$

Well-Founded Recursion and Induction

- Let U be a type and (U, R) be well founded.
- The well-founded induction principle for (U, R) is:

  $$(\forall x : U \bullet (\forall y : U \bullet y \; R \; x \Rightarrow P \, y) \Rightarrow P \, x) \\ \Rightarrow (\forall x : U \bullet P \, x)$$

  holds for every property P of U.
- Two important special cases of well-founded induction:
  - Structural induction.
  - Transfinite induction.
- A function f can be defined by well-founded recursion.
  - Each recursive application of f must be applied to at least one argument that is smaller with respect to R.

Example 1: Nat as a Well-Founded Structure

- $(\mathsf{Nat}, <)$ is well-founded where $m < n$ means

  $$\exists\, k : \mathsf{Nat} \bullet k \neq 0 \wedge m + k = n.$$

- The well-founded induction principle for Nat is:

  $$(\forall\, x : \mathsf{Nat} \bullet (\forall\, y : \mathsf{Nat} \bullet y < x \Rightarrow P\, y) \Rightarrow P\, x)$$
  $$\Rightarrow (\forall\, x : \mathsf{Nat} \bullet P\, x)$$

  holds for every property P of Nat. This principle is called strong induction, complete induction, or course-of-values induction.

- Theorem. The following are equivalent:

  1. Structural induction for Nat (weak induction).
  2. Well-founded induction for Nat (strong induction).

Example 1: Functions Defined by WFR

- Addition ($+ : \mathsf{Nat} \times \mathsf{Nat} \to \mathsf{Nat}$) is defined by well-founded recursion as:
  1. $x + 0 = x$.
  2. $x + S\,y = S\,(x + y)$. Note that $y < S\,y$ holds.
- Multiplication ($* : \mathsf{Nat} \times \mathsf{Nat} \to \mathsf{Nat}$) is defined by well-founded recursion as:
  1. $x * 0 = 0$.
  2. $x * S\,y = (x * y) + x$. Note that $y < S\,y$ holds.
- The function $\mathsf{fib} : \mathsf{Nat} \to \mathsf{Nat}$ that maps n to the nth Fibonacci number is defined by pattern matching as:
  1. $\mathsf{fib}\,0 = 0$.
  2. $\mathsf{fib}\,S\,0 = S\,0$.
  3. $\mathsf{fib}\,S\,(S\,x) = \mathsf{fib}\,(S\,x) + \mathsf{fib}\,x$.

  Note that $x, S\,x < S\,(S\,x)$.

Example 2: BinTree as a Well-Founded Structure

- $(\mathsf{BinTree}, <)$ is well-founded where $t_1 < t_2$ means t_1 is a proper subtree of t_2.
- The well-founded induction principle for BinTree is:

$$(\forall\, x : \mathsf{BinTree} \bullet (\forall\, y : \mathsf{BinTree} \bullet y < x \Rightarrow P\, y) \Rightarrow P\, x)$$
$$\Rightarrow (\forall\, x : \mathsf{BinTree} \bullet P\, x)$$

  holds for every property P of BinTree.
- Theorem. The following are equivalent:
  1. Structural induction for BinTree.
  2. Well-founded induction for BinTree.

Weak Induction vs. Strong Induction

- Weak induction:

  $$(P\,0 \wedge (\forall\, x : \mathrm{Nat} \bullet P\,x \Rightarrow P\,(S\,x))) \Rightarrow (\forall\, x : \mathrm{Nat} \bullet P\,x).$$

  Form of Proof:

  1. Base case: Show $P\,0$.
  2. Induction step: Assume $P\,x$. Show $P\,(S\,x)$.

- Strong induction:

  $$(\forall\, x : \mathrm{Nat} \bullet (\forall\, y : \mathrm{Nat} \bullet y < x \Rightarrow P\,y) \Rightarrow P\,x)$$
  $$\Rightarrow (\forall\, x : \mathrm{Nat} \bullet P\,x).$$

  Form of Proof:

  1. Base case: Let $x = 0$. (Assume nothing.) Show $P\,x$.
  2. Induction step: Let $x > 0$. Assume $P\,y$ for all $y < x$. Show $P\,x$.

- Strong induction provides a stronger induction hypothesis than weak induction.