CAS 701
Logic and Discrete Mathematics
Fall 2017

2 Recursion and Induction

William M. Farmer

Department of Computing and Software
McMaster University

October 6, 2017

What are Recursion and Induction?

- Recursion is a method of defining a structure or operation in terms of itself.
  - One of the most fundamental ideas of computing.
  - Can make some specifications, descriptions, and programs easier to express, understand, and prove correct.
- Induction is a method of proof based on a recursively defined structure.
  - The recursively defined structure and the proof method are specified by an induction principle.
  - Induction is especially useful for proving properties about recursively defined operations.
- The terms "recursion" and "induction" are often used interchangeably.

Example: Natural Numbers

- Recursive definition of $\mathbb{N}$:
  1. $0 \in \mathbb{N}$.
  2. If $n \in \mathbb{N}$, then $S(n) \in \mathbb{N}$.
  3. The members of $\mathbb{N}$ are distinct (no confusion).
  4. $\mathbb{N}$ is the smallest such set (no junk).
- Induction principle for $\mathbb{N}$ (called mathematical induction):

  $$\forall\, P : \mathbb{N} \rightarrow \mathrm{Bool}\ .\ [P(0) \wedge (\forall\, n \in \mathbb{N}\ .\ P(n) \Rightarrow P(S(n)))] \Rightarrow \forall\, n \in \mathbb{N}\ .\ P(n)$$

- Alternate induction principle for $\mathbb{N}$ (strong mathematical induction):

  $$\forall\, P : \mathbb{N} \rightarrow \mathrm{Bool}\ .\ [\forall\, n \in \mathbb{N}\ .\ (\forall\, m \in \mathbb{N}\ .\ m < n \Rightarrow P(m)) \Rightarrow P(n)] \Rightarrow \forall\, n \in \mathbb{N}\ .\ P(n)$$

Example: Stacks of Natural Numbers

- Recursive definition of Stack:
  1. bottom $\in$ Stack.
  2. If $n \in \mathbb{N}$ and $s \in$ Stack, then push$(n, s) \in$ Stack.
  3. The members of Stack are distinct (no confusion).
  4. Stack is the smallest such set (no junk).
- Induction principle for Stack:

$$
\begin{array}{l}
\forall\, P : \mathsf{Stack} \to \mathsf{Bool}\ . \\
\quad [P(\mathsf{bottom}) \wedge (\forall\, s \in \mathsf{Stack}\ .\ P(s) \Rightarrow \\
\quad\quad (\forall\, n \in \mathbb{N}\ .\ P(\mathsf{push}(n, s)))] \Rightarrow \forall\, s \in \mathsf{Stack}\ .\ P(s)
\end{array}
$$

Transfinite Induction

- Transfinite induction principle:

$$\begin{array}{l} \forall\, P : \mathbb{O} \to \mathsf{Bool}\ . \\ \quad [\forall\, \alpha \in \mathbb{O}\ .\ (\forall\, \beta \in \mathbb{O}\ .\ \beta < \alpha \Rightarrow P(\beta)) \Rightarrow P(\alpha)] \\ \qquad \Rightarrow \forall\, \alpha \in \mathbb{O}\ .\ P(\alpha) \end{array}$$

- Induction principle for $\gamma \in \mathbb{O}$:

$$\begin{array}{l} \forall\, P : \mathbb{O}_\gamma \to \mathsf{Bool}\ . \\ \quad [\forall\, \alpha \in \mathbb{O}_\gamma\ .\ (\forall\, \beta \in \mathbb{O}_\gamma\ .\ \beta < \alpha \Rightarrow P(\beta)) \Rightarrow P(\alpha)] \\ \qquad \Rightarrow \forall\, \alpha \in \mathbb{O}_\gamma\ .\ P(\alpha) \end{array}$$

  where $\mathbb{O}_\gamma = \{\alpha \in \mathbb{O} \mid \alpha < \gamma\}$.

  - Strong mathematical induction is just the induction principle for ω.
  - Nested induction arguments can be expressed using the induction principle for a suitable ordinal.

Strengthening the Induction Hypothesis

- Sometimes a statement cannot be proved by induction because the the resulting induction hypothesis is too weak.
- The strategy of strengthening the induction hypothesis is to prove a stronger statement that results in a stronger induction hypothesis.
- Example: Every square number is the sum of two triangle numbers.

Recursive Function Definitions

- Recursion is extremely useful for defining functions.
  - Can facilitate both reasoning and computation.
- A faulty recursive definition may lead to inconsistencies.
  - Example: $\forall\, n : \mathbb{N}\, .\, f(n) = f(n) + 1$.
- There are several schemes for defining functions by recursion.

Recursive Definition Schemes

- Each scheme has a set of instance requirements.
- A scheme is proper if every instance of the scheme actually defines a function.
- The domain of a scheme is the set of functions f such that f is definable by some instance of the scheme.
- Designers of mechanized mathematics systems prefer schemes which:
  - Are proper.
  - Have easily checked instance requirements.
  - Have a large domain of useful functions.

The Primitive Recursive Functions (1/2)

- The class $\mathcal{P}$ of primitive recursive functions is the smallest set of $f : \mathbb{N} \times \cdots \times \mathbb{N} \to \mathbb{N}$ closed under the following rules:
  1. Successor Function $(\lambda\, x : \mathbb{N} \,.\, x + 1) \in \mathcal{P}$.
  2. Constant Functions Each $(\lambda\, x_1, \ldots, x_n : \mathbb{N} \,.\, m) \in \mathcal{P}$ where $0 \leq m, n$.
  3. Projection Functions Each $(\lambda\, x_1, \ldots, x_n : \mathbb{N} \,.\, x_i) \in \mathcal{P}$ where $1 \leq n$ and $1 \leq i \leq n$.
  4. Composition If $g_1, \ldots, g_m, h \in \mathcal{P}$, then $f \in \mathcal{P}$ where:
     $$\begin{array}{l} \forall\, x_1, \ldots, x_n : \mathbb{N} \,. \\ \quad f(x_1, \ldots, x_n) = h(g_1(x_1, \ldots, x_n), \ldots, g_m(x_1, \ldots, x_n)). \end{array}$$
  5. Primitive Recursion If $g, h \in \mathcal{P}$, then $f \in \mathcal{P}$ where:
     $$\begin{array}{l} \forall\, x_2, \ldots, x_n : \mathbb{N} \,.\, f(0, x_2, \ldots, x_n) = g(x_2, \ldots, x_n). \\ \forall\, x_1, \ldots, x_n : \mathbb{N} \,. \\ \quad f(x_1 + 1, x_2, \ldots, x_n) = \\ \qquad h(x_1, f(x_1, x_2, \ldots, x_n), x_2, \ldots, x_n). \end{array}$$

The Primitive Recursive Functions (2/2)

- Example. The factorial function $f : \mathbb{N} \to \mathbb{N}$ is defined by:
  1. $f(0) = g(\,) = 1$.
  2. $f(n+1) = h(n, f(n))$ where $h(x, y) = y * (x+1)$.
- The primitive recursion scheme is proper.
- $\mathcal{P}$ is a very large, but proper, subset of the total computable functions on $\mathbb{N}$.
  - $\mathcal{P}$ contains almost all functions on $\mathbb{N}$ commonly found in mathematics.
- Theorem. There exists a total computable function $f : \mathbb{N} \to \mathbb{N}$ such that $f \notin \mathcal{P}$.

  Proof: Construct f by diagonalization.

Ackermann Function

- A leading version of the Ackermann function $A : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ is recursively defined by:

  $$A(0, n) = n + 1.$$
  $$A(m, 0) = A(m - 1, 1) \text{ if } m > 0.$$
  $$A(m, n) = A(m - 1, A(m, n - 1)) \text{ if } m, n > 0.$$

- Theorem.

  1. A is a total computable function.
  2. A is not primitive recursive.

- A grows rapidly at an extreme rate!

Well-Founded Relations

- A relation $R \subseteq A \times A$ is well-founded, if for all nonempty $B \subseteq A$, there is some $a \in B$ such that, for all $b \in B$, $\neg(b\ R\ a)$.
  - a is called an R-least element of B.
- Proposition. If R is a strict total order, then R is well-founded iff R is a well-order.
- Induction principle for a well-founded relation $R \subseteq A \times A$:

$$\begin{array}{l} \forall\, P : A \to \mathrm{Bool}\ . \\ \quad [\forall\, x : A\ .\ (\forall\, y : A\ .\ y\ R\ x \Rightarrow P(y)) \Rightarrow P(x)] \\ \qquad \Rightarrow \forall\, x : A\ .\ P(x) \end{array}$$

Well-Founded Recursion

- A definition via well-founded recursion is a tuple $W = (T, f, D, R)$ where
  - $T = (L, \Gamma)$ is a theory.
  - f is a constant of type $\alpha \rightarrow \alpha$ not in L.
  - D is a sentence of the form
    $$\forall x \,.\, f(x) = E(f(a_1(x)), \ldots, f(a_k(x))).$$
  - R is a well-founded relation on α.
- W defines f to be a total function in T by well-founded recursion if:
  1. $T \models \forall x \,.\, R\text{-least}(x) \Rightarrow E(f(a_1(x)), \ldots, f(a_k(x))) = t$ for some term t of L.
  2. $T \models \forall x \,.\, \neg R\text{-least}(x) \Rightarrow a_1(x) \; R \; x \wedge \cdots \wedge a_k(x) \; R \; x$.
- The definitional extension resulting from W is the theory $(L \cup \{f\}, \Gamma \cup \{D\})$.

Example

- Let $W = (P, f, D, <)$ where
  - P is first-order Peano arithmetic.
  - $f : \mathbb{N} \rightarrow \mathbb{N}$..
  - D is

$$\forall\, n \,.\, f(n) = \mathrm{if}(n = 0, 1, f(n-1) * n).$$

  - $<$ is the usual order on $\mathbb{N}$.
- The W defines the factorial function in P.

Structural Recursion and Induction

- Structural recursion is a disguised form of well-founded recursion in which the well-founded relation is a less-structure to more-structure relationship.
- Examples of sets defined by structural recursion:
  - Inductive types such as stacks, lists, and trees.
  - Formal languages such as programming languages, the terms of FOL, and the formulas of FOL.
- Structural induction is induction over a set defined by structural recursion.
- Structural induction principle: A property P holds for all members of a set S defined by structural recursion if:
  1. P holds for all members of S having minimal structure.
  2. P holds for a structural combination of members of S whenever it holds for the members themselves.

Inductive Types

- An inductive type is a type t whose members are specified by a finite set of constructors (where $m_1, \ldots, m_n \geq 0$):

$$C_1 : t^1_1 \rightarrow \cdots \rightarrow t^1_{m_1}.$$
$$\vdots$$
$$C_n : t^n_1 \rightarrow \cdots \rightarrow t^n_{m_n}.$$

- The type t may be included in the types $t^1_1, \ldots, t^n_{m_n}$.
- The members of the inductive type are exactly the expressions that can be built from the constructors $C_1, \ldots, C_n$.
  - An inductive type thus defines a language.
  - The members of the type are effectively literals.
- A inductive type definition induces a structural induction principle.
- Functions on an inductive type can be defined by pattern matching.

Examples of Inductive Types

- ```
  Inductive Bool =
     | true : Bool
     | false : Bool
  ```
- ```
  Inductive Nat =
     | zero : Nat
     | suc : Nat -> Nat
  ```
- ```
  Inductive Stack =
     | bottom : Stack
     | push : Nat -> Stack -> Stack
  ```
- ```
  Inductive List =
     | nil : List
     | cons : Nat -> List -> List
  ```
- ```
  Inductive BinTree =
     | leaf : Nat -> BinTree
     | branch : BinTree -> BinTree -> BinTree
  ```
  ```

Fixed Point Operators

- A fixed-point operator on a set $\mathcal{F}$ of higher-order functions is a function fix such that, for all $f \in \mathcal{F}$, $\mathrm{fix}(f) = f(\mathrm{fix}(f))$.
- Fixed point operators offer an alternative way to define a recursive function.
  - The equation
    $$\mathrm{fix}(f)(x_1, \ldots, x_n) = f(\mathrm{fix}(f))(x_1, \ldots, x_n)$$
    is a recursive definition.
  - n applications of f to $\mathrm{fix}(f)$ unfolds the recursive definition n times.
- Example: The Y combinator, defined as
  $$\lambda f \,.\, (\lambda x \,.\, f(x\,x))(\lambda x \,.\, f(x\,x)),$$
  is a fixed-point operator on lambda expressions.

Example: Monotone Functionals

- A functional is an expression of type $\alpha \rightharpoonup \alpha$ where $\alpha = \alpha_1 \times \cdots \times \alpha_n \rightharpoonup \alpha_{n+1}$.
- Subfunction:
  $\forall\, g, h : \alpha \;.\; g \sqsubseteq_\alpha h \Leftrightarrow$
  $\quad \forall\, x_1 : \alpha_1, \ldots, x_n : \alpha_n \;.\; g(x_1, \ldots, x_n)\!\downarrow \Rightarrow$
  $\quad g(x_1, \ldots, x_n) = h(x_1, \ldots, x_n)$.
- Monotone:
  $\forall\, F : \alpha \rightharpoonup \alpha \;.\; \mathrm{monotone}_\alpha(F) \Leftrightarrow$
  $\quad \forall\, g, h : \alpha \;.\; g \sqsubseteq_\alpha h \Rightarrow F(g) \sqsubseteq_\alpha F(h)$.
- Fixed Point Theorem. Every monotone functional has a least fixed point.

  Proof: $F^\gamma(\triangle_\alpha)$ must be a fixed point for some ordinal γ, where $\triangle_\alpha$ is the empty function of type α.

Monotone Functional Recursion

- A recursive definition via a monotone functional is a triple $M = (T, f, F)$ where:
  - $T = (L, \Gamma)$ is a theory (in a higher-order logic that admits partial functions).
  - f is a new constant of type α not in L.
  - F is a functional of type $\alpha \rightharpoonup \alpha$ which is monotone in T.
- The defining axiom of M is D which says "f is a least fixed point of F".
- The definitional extension resulting from M is the theory $(L \cup \{f\}, \Gamma \cup \{D\})$.

Examples

- Empty function:
  $\lambda f : \mathbb{Z} \rightharpoonup \mathbb{Z} \,.\, \lambda n : \mathbb{Z} \,.\, f(n).$
- Empty function:
  $\lambda f : \mathbb{Z} \rightharpoonup \mathbb{Z} \,.\, \lambda n : \mathbb{Z} \,.\, f(n) + 1.$
- Factorial:
  $\lambda f : \mathbb{N} \rightharpoonup \mathbb{N} \,.\, \lambda n : \mathbb{N} \,.\, \mathrm{if}(n = 0, 1, f(n-1) * n).$
- Sum:
  $$\begin{aligned}&\lambda \sigma : \mathbb{Z} \times \mathbb{Z} \times (\mathbb{Z} \rightharpoonup \mathbb{R}) \rightharpoonup \mathbb{R} \,.\\ &\quad \lambda m, n : \mathbb{Z}, f : \mathbb{Z} \rightharpoonup \mathbb{R} \,.\\ &\qquad \mathrm{if}(m \leq n, \sigma(m, n-1, f) + f(n), 0).\end{aligned}$$