CAS 701 Fall 2008

07 Recursion and Induction

William M. Farmer

Department of Computing and Software
McMaster University

26 November 2008

McMaster University

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

$$\begin{array}{l} \forall\, P : \mathbf{N} \rightarrow * \;. \\ \qquad [P(0) \wedge (\forall\, x : \mathbf{N} \;.\; P(x) \Rightarrow P(S(x)))] \\ \qquad\qquad \Rightarrow \\ \qquad \forall\, x : \mathbf{N} \;.\; P(x) \end{array}$$

- This induction principle is also called mathematical induction.

Example: Stacks of Natural Numbers

- Recursive definition of Stack:
  1. Bottom $\in$ Stack.
  2. If $n \in \mathbf{N}$ 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:

$$
\begin{array}{l}
\forall\, P : \mathrm{Stack} \rightarrow * \;. \\
\qquad [P(\mathrm{Bottom}) \wedge \\
\qquad\; (\forall\, s : \mathrm{Stack} \;.\; P(s) \Rightarrow (\forall\, n : \mathbf{N} \;.\; P(\mathrm{Push}(n, s))))] \\
\qquad\qquad \Rightarrow \\
\qquad \forall\, s : \mathrm{Stack} \;.\; P(s)
\end{array}
$$

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: $\forall\, n : \mathbf{N}\ .\ f(n) = f(n) + 1$.
- 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 $f : \mathbf{N} \times \cdots \times \mathbf{N} \rightarrow \mathbf{N}$ closed under the following rules:
  1. Successor Function $(\lambda\, x : \mathbf{N}\, .\, x + 1) \in \mathcal{P}$.
  2. Constant Functions Each $(\lambda\, x_1, \ldots, x_n : \mathbf{N}\, .\, m) \in \mathcal{P}$ where $0 \leq m, n$.
  3. Projection Functions Each $(\lambda\, x_1, \ldots, x_n : \mathbf{N}\, .\, x_i) \in \mathcal{P}$ 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:

     $$\begin{array}{l} \forall\, x_1, \ldots, x_n : \mathbf{N}\, . \\ \quad f(x_1, \ldots, x_n) = h(g_1(x_1, \ldots, x_n), \ldots, g_m(x_1, \ldots, x_n)). \end{array}$$

  5. Primitive Recursion If $g, h \in \mathcal{P}$, then $f \in \mathcal{P}$ where:

     $$\begin{array}{l} \forall\, x_2, \ldots, x_n : \mathbf{N}\, .\, f(0, x_2, \ldots, x_n) = g(x_2, \ldots, x_n). \\ \forall\, x_1, \ldots, x_n : \mathbf{N}\, . \\ \quad f(x_1 + 1, x_2, \ldots, x_n) = \\ \quad\quad h(x_1, f(x_1, x_2, \ldots, x_n), x_2, \ldots, x_n). \end{array}$$

The Primitive Recursive Functions (2/2)

- Example. The factorial function $f : \mathbf{N} \to \mathbf{N}$ 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 computable total functions on $\mathbf{N}$.
  - $\mathcal{P}$ contains almost all functions on $\mathbf{N}$ commonly found in mathematics.
- Theorem. There exists a computable total function $f : \mathbf{N} \to \mathbf{N}$ such that $f \notin \mathcal{P}$.

  Proof: Construct f by diagonalization.

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.

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 $\alpha \to \alpha$ 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.
  - $f : \mathbf{N} \rightarrow \mathbf{N}$..
  - D is

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

  - $<$ is the usual order on $\mathbf{N}$.
- 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 data types such as lists, trees, and stacks.
  - 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.

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:
  $\forall\, F : \alpha \rightharpoonup \alpha \,.\, \mathsf{monotone}_\alpha(F) \Leftrightarrow$
  $\quad \forall\, g, h : \alpha \,.\, g \sqsubseteq_\alpha h \Rightarrow F(g) \sqsubseteq_\alpha F(h).$
- 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 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:
  $\lambda f : \mathbf{Z} \rightharpoonup \mathbf{Z} \,.\, \lambda n : \mathbf{Z} \,.\, f(n).$
- Empty function:
  $\lambda f : \mathbf{Z} \rightharpoonup \mathbf{Z} \,.\, \lambda n : \mathbf{Z} \,.\, f(n) + 1.$
- Factorial:
  $\lambda f : \mathbf{N} \rightharpoonup \mathbf{N} \,.\, \lambda n : \mathbf{N} \,.\, \mathrm{if}(n = 0, 1, f(n-1) * n).$
- Sum:
  $\lambda \sigma : \mathbf{Z} \times \mathbf{Z} \times (\mathbf{Z} \rightharpoonup \mathbf{R}) \rightharpoonup \mathbf{R} \,.$
  $\quad \lambda m, n : \mathbf{Z}, f : \mathbf{Z} \rightharpoonup \mathbf{R} \,.$
  $\quad\quad \mathrm{if}(m \leq n, \sigma(m, n-1, f) + f(n), 0).$