Inductive Sets and Recursion

CS496

Inductively Specified Set

- We'll see an informal introduction
- References
  - A didactic reference is:
    Mathematical Foundations of Computer Science, Sets, Relations, and Induction, Peter A. Fejer and Dan A. Simovici, Texts and Monographs in Computer Science, Springer Verlag, 1991
  - A standard reference is (but more challenging):
    An Introduction to Inductive Definitions, Peter Aczel, Studies in Logic and the Foundations of Mathematics Volume 90, 1977, Pages 739-782

Inductively Specified Set

- A means of defining sets that
  1. Describes how to generate is elements
     - Derivations
  2. Comes equipped with a technique for proving properties of its elements
     - Structural Induction
  3. Comes equipped with a technique for defining functions over its elements
     - Structural Recursion

Specifying an Inductive Definition

All inductive definitions require specifying two elements

1. A universe
   - In PL the universe is typically specified by giving an alphabet Σ and then taking the universe to be the set of all words from that alphabet
2. The smallest subset of the universe that satisfies certain conditions
   - This set is therefore a subset of the words in Σ

An Example of A Universe

Let Σ be the set of symbols

$$(\qquad) \qquad z \qquad s$$

The set of words over Σ, denoted Σ^*, consists of

$$\{z, s, zz, sz, zs, ss, zsss, s, s((, (()), \ldots\}$$

A First Example of an Inductive Definition

- We already specified the universe in the previous slide
- Now lets specify the inductive set proper

Example of inductive definition

Let S be the smallest subset of Σ^* that satisfies:

1. $z \in S$,
2. $s(n) \in S$ whenever $n \in S$.

- The first clause is called the base clause or rule
- The second clause is called the inductive clause or rule

A First Example (cont.)

Let S be the smallest subset of Σ^* that satisfies:

1. $z \in S$,
2. $s(n) \in S$ whenever $n \in S$.

What sets satisfy the specification?

A First Example (cont.)

Let S be the smallest subset of Σ^* that satisfies:

1. $z \in S$,
2. $s(n) \in S$ whenever $n \in S$.

What sets satisfy the specification?

- $\{z, s(z), s(s(z)), s(s(s(z))), \ldots\}$
- $\{z, s(z), s(s(z)), s(s(s(z))), \ldots\} \cup \{s, s(s), s(s(s)), \ldots\}$

Smallest implies:

- Exactly those elements generated by the specification
- We can give a derivation showing why each element belongs in the set.

Derivation of Set Elements

Let S be the smallest subset of Σ^* satisfying

1. $z \in S$,
2. $s(z) \in S$ whenever $n \in S$.

Example: $s(s(s(z)))$

- $z \in S$ (by rule 1)
- $s(z) \in S$ (by rule 2)
- $s(s(z)) \in S$ (by rule 2)
- $s(s(s(z))) \in S$ (by rule 2)

Non-example: zs

Example: Primary Colors

- Let Σ be the English alphabet

Primary Colors defined inductively

Let $PCol$ be the smallest subset of Σ^* that satisfies:

1. $Red \in PCol$
2. $Green \in PCol$
3. $Blue \in PCol$

- This definition only has base clauses
- It defines a finite set, namely $\{Red, Green, Blue\}$

Simplifying the Definition of Inductive Sets – Dropping the Universe

- As mentioned, Σ^* below is the known as the universe

  Let S be the smallest subset of Σ^* satisfying

  1. $z \in S$,
  2. $s(z) \in S$ whenever $n \in S$.

- We often drop the reference to the universe

  Let S be the smallest set satisfying

  1. $z \in S$,
  2. $s(z) \in S$ whenever $n \in S$.

- It is mathematically less precise, but sufficiently precise for our programming examples

Alternative Notations for Defining Inductive Sets

We'll briefly introduce three alternative notations for defining inductive sets

1. Prose (already seen) notation
2. Rule notation
3. BNF notation

For each we will exemplify with the set of natural numbers and a derivation that $s(s(z))$ belongs to the set

Notation 1 – Prose

Sample definition

Let S be the smallest set that satisfies:

1. $z \in S$,
2. $s(n) \in S$ whenever $n \in S$.

Sample derivation

- $z \in S$ (by rule 1)
- $s(z) \in S$ (by rule 2)
- $s(s(z)) \in S$ (by rule 2)

Notation 2 – Rule Notation

Sample definition

$$\frac{}{z \in S}\,\mathit{Rule}\ 1 \qquad \frac{n \in S}{s(n) \in S}\,\mathit{Rule}\ 2$$

Sample derivation

$$\cfrac{\cfrac{\cfrac{}{z \in S}\,\mathit{Rule}\ 1}{s(z) \in S}\,\mathit{Rule}\ 2}{s(s(z)) \in S}\,\mathit{Rule}\ 2$$

Notation 3 – BNF or Grammar Notation

Sample definition

$$\begin{array}{rcl} \langle S \rangle & ::= & z \\ \langle S \rangle & ::= & s(\langle S \rangle) \end{array}$$

- $\langle S \rangle$ is called a non-terminal
- $z, s, ($ and $)$ are called terminals
- This definition can be abbreviated

$$\langle S \rangle \quad ::= \quad z \mid s(\langle S \rangle)$$

Sample derivation

$$\begin{array}{rcl} \langle S \rangle & \Rightarrow & s(\langle S \rangle) \\ & \Rightarrow & s(s(\langle S \rangle)) \\ & \Rightarrow & s(s(z)) \end{array}$$

Primary Colors in Rule Notation

$$\frac{}{Red \in PCol} \qquad \frac{}{Green \in PCol}$$

$$\frac{}{Blue \in PCol}$$

Examples of elements of $PCol$

- Red
- $Green$

Another example: Lists (over a set S)

$$\frac{}{nil \in List(S)}$$

$$\frac{s \in S \quad l \in List(S)}{cons(s, l) \in List(S)}$$

Examples of elements of $List(\mathbb{N})$

- nil
- $cons(4, nil)$
- $cons(1, cons(2, cons(5, cons(0, nil))))$

Another inductive set: Trees (over a set S)

$$\frac{s \in S}{leaf(s) \in BTree(S)}$$

$$\frac{l \in BTree(S) \quad r \in BTree(S)}{node(l, r) \in BTree(S)}$$

Example of elements in $Btree(\mathbb{N})$

- $leaf(2)$
- $node(leaf(2), leaf(3))$
- $node(node(leaf(2), node(leaf(7), leaf(2))), node(leaf(2), leaf(1)))$

Defining functions over inductive sets

- Structural recursion: technique for defining functions over inductive sets S
- When defining f over an inductive set S return:
  - Known values, for s in S justified by base rules
  - Composition of known values and f applied to the parts that conform s, for s in S justified by inductive rules

Example

Let S be the subset of Σ^* satisfying

1. $z \in S$,
2. $s(z) \in S$ whenever $n \in S$.

$$noOfSuc :: S \to \mathbb{N}$$

$$\begin{array}{lcl} noOfSuc(z) & = & 0 \\ noOfSuc(s(n)) & = & 1 + noOfSuc(n) \end{array}$$

Simple recursive functions over $\mathit{List}(\mathbb{Z})$

$\mathit{sizeL} :: \mathit{List}(\mathbb{N}) \to \mathbb{N}$

$$\begin{aligned} \mathit{sizeL}(\mathit{nil}) &= 0 \\ \mathit{sizeL}(\mathit{cons}(n, l)) &= 1 + \mathit{sizeL}(l) \end{aligned}$$

$\mathit{sumL} :: \mathit{List}(\mathbb{N}) \to \mathbb{N}$

$$\begin{aligned} \mathit{sumL}(\mathit{nil}) &= 0 \\ \mathit{sumL}(\mathit{cons}(n, l)) &= n + \mathit{sumL}(l) \end{aligned}$$

Recursive Functions over Trees of Numbers

$$\frac{n \in S}{\mathit{leaf}(n) \in \mathit{BTree}(S)}$$

$$\frac{l \in \mathit{BTree}(S) \quad r \in \mathit{BTree}(S)}{\mathit{node}(l, r) \in \mathit{BTree}(S)}$$

$$\begin{array}{lcl}
\multicolumn{3}{l}{\mathit{noOfNodes} :: \mathit{Tree}(\mathbb{N}) \to \mathbb{N}} \\
\mathit{noOfNodes}(\mathit{leaf}(n)) & = & 1 \\
\mathit{noOfNodes}(\mathit{node}(l, r)) & = & 1 + \mathit{noOfNodes}(l) + \mathit{noOfNodes}(r)
\end{array}$$

Recursive Functions over Trees of Numbers

$$\frac{n \in S}{\mathit{leaf}(n) \in \mathit{BTree}(S)}$$

$$\frac{l \in \mathit{BTree}(S) \quad r \in \mathit{BTree}(S)}{\mathit{node}(l, r) \in \mathit{BTree}(S)}$$

$$\begin{array}{lcl}
\mathit{incTree} :: \mathit{Tree}(\mathbb{N}) \rightarrow \mathit{Tree}(\mathbb{N}) & & \\
\mathit{incTree}(\mathit{leaf}(n)) & = & \mathit{leaf}(n+1) \\
\mathit{incTree}(\mathit{node}(l, r)) & = & \mathit{node}(\mathit{incTree}(l), \mathit{incTree}(r))
\end{array}$$

Inductive Sets

Defining Functions over Inductive Sets

Representing Inductive Sets in OCaml

Proving Properties of Elements of Inductive Sets

Representing the set $List(\mathbb{Z})$ in OCaml

Inductive Set (Maths)

$$\frac{}{nil \in List(\mathbb{Z})} \qquad \frac{s \in \mathbb{Z} \quad l \in List(\mathbb{Z})}{cons(s, l) \in List(\mathbb{Z})}$$

Encoding in OCaml (PL)

```
type list_int = Nil | Cons of int*list_int
```

The OCaml expression

```
Cons(1,Cons(2,Cons(3,Nil)))
```

represents the list `cons(1,cons(2,cons(3,nil)))`

Representing the set $List(\mathbb{Z})$ in OCaml

```
type list_int = Nil | Cons of int*list_int
```

- `list_nat` is an example of an Algebraic Data Type
  - Name convention: initial lower case; underscores for multiword names
- `Nil` and `Cons` are called Constructors
  - Name convention: initial upper case; use camel notation (eg. EmptyStack)
- Constructors are not functions

```
# Cons;;
Error: The constructor Cons expects 2 argument(s),
but is applied here to 0 argument(s)
```

Trees of Numbers in OCaml

Inductive Set (Maths)

$$\frac{n \in \mathbb{Z}}{\mathit{leaf}(n) \in \mathit{BTree}(\mathbb{Z})} \qquad \frac{l \in \mathit{BTree}(\mathbb{Z}) \quad r \in \mathit{BTree}(\mathbb{Z})}{\mathit{node}(l, r) \in \mathit{BTree}(\mathbb{Z})}$$

Encoding in OCaml (PL)

```
type bTree = Leaf of int | Node of bTree*bTree
```

The OCaml expression

```
Node(Node(Leaf 2,Leaf 2),
     Node(Leaf 5,Node(Leaf 7,Leaf 8)))
```

encodes the tree
$\mathit{node}(\mathit{node}(\mathit{leaf}(2), \mathit{leaf}(2)), \mathit{node}(\mathit{leaf}(5), \mathit{node}(\mathit{leaf}(7), \mathit{leaf}(8))))$

Polymorphic Containers

- `option` type (built-in)

```
type 'a option = None | Some of 'a
```

- Disjoint union

```
type ('a, 'b) either = Left of 'a | Right of 'b
```

- Polymorphic lists

```
type 'a list = Nil | Cons of 'a*'a list
```

- Polymorphic trees

```
type ('a, 'b) ab_tree =
  | Leaf of 'a
  | Node of 'b*('a, 'b) ab_tree*('a, 'b) ab_tree
```

Recursive Functions over Inductive Sets in OCaml

Computing the sum of a list in OCaml

```
type listNat = Nil | Cons of int*listNat
```

```
let rec list_sum = function
  | Nil -> 0
  | Cons(h,t) -> h + list_sum t
```

Key points:

- recursion occurs in procedure exactly where recursion occurs in BNF
- we may assume procedure "works" for sub-structures of the same type

More Examples

Add one to each element:

```
# list_inc [];;
[]
# list_inc [1];;
[2]
# list_inc [1;2;3];;
[2;3;4]
```

Append:

```
# list_app [1;2;3] [4;5]
[1;2;3;4;5]
# list_app [] [4;5]
[4;5]
```

More Examples of Recursive Functions

```
let rec list_inc = function
 | Nil -> []
 | Node(h,t) -> Cons(h+1,list_rec t)

let rec list-app l1 l2 =
    match l1 with
      | Nil -> l2
      | Node(h,t) -> Cons(h,list_app t l2)
```

Trees of Numbers $BTree(\mathbb{N})$ in OCaml

```
type bTree = Leaf of int | Node of bTree*bTree
```

Example:

```
let rec tree_sum = function
    | Leaf n -> Leaf (n+1)
    | Node(l,r) -> Node(tree_sum l,tree_sum r)
```

```
# tree_sum (Node(Node(Leaf(2),Leaf(3)),
                Node(Leaf(1),Node(Leaf(4),Leaf(5)))))
15
```

Tree Examples

```
let rec tree-flip = function
 | Leaf n -> Leaf n
 | Node(l,r) -> Node(tree_flip r,tree_flip l)
```

```
# tree_flip (Node(Node(Leaf(2),Leaf(3)),
                Node(Leaf(1),Node(Leaf(4),Leaf(5)))))
Node(Node(Node(Leaf(5),Leaf(4)),Leaf(1)),
      Node(Leaf(3),Leaf(2)))
```

Inductive Sets

Defining Functions over Inductive Sets

Representing Inductive Sets in OCaml

Proving Properties of Elements of Inductive Sets

Proof by Structural Induction

S is an inductive set and P is a property of its elements

- How to prove
$$\forall x \in S. P(x)$$
- Resort to Structural Induction:
    1. Prove P is true on simple structures (base rules).
    2. Prove that, if P is true on the substructures of x (Induction Hypothesis), then it is true on x itself (inductive rules).

Example of Proof using Structural Induction

$$\frac{n \in \mathbb{N}}{\mathit{leaf}(n) \in \mathit{BTree}(\mathbb{N})} \qquad \frac{l \in \mathit{BTree}(\mathbb{N}) \quad r \in \mathit{BTree}(\mathbb{N})}{\mathit{node}(l, r) \in \mathit{BTree}(\mathbb{N})}$$

Consider

$$P(t) = \text{“}t \text{ contains an odd number of nodes”}$$

- Aim: prove $\forall t \in \mathit{BTree}(\mathbb{N}).P(t)$
- Tool: use Structural Induction

Example of Proof using Structural Induction (cont.)

$$\frac{n \in \mathbb{N}}{leaf(n) \in BTree(\mathbb{N})} \qquad \frac{l \in BTree(\mathbb{N}) \quad r \in BTree(\mathbb{N})}{node(l, r) \in BTree(\mathbb{N})}$$

Consider

$$P(t) = \text{“}t \text{ contains an odd number of nodes”}$$

- Base case:
  - $t = leaf(i)$, where i is a number.
  - Reasoning: $P(t)$ holds immediately since a leaf is a node and 1 is odd.
- Inductive case: (next slide)

Example of Proof using Structural Induction (cont.)

$$\frac{n \in \mathbb{N}}{\mathit{leaf}(n) \in \mathit{BTree}(\mathbb{N})} \qquad \frac{l \in \mathit{BTree}(\mathbb{N}) \quad r \in \mathit{BTree}(\mathbb{N})}{\mathit{node}(l, r) \in \mathit{BTree}(\mathbb{N})}$$

Consider

$$P(t) = \text{“}t \text{ contains an odd number of nodes”}$$

- Inductive case:
  - $t = \mathit{node}(t_1, t_2)$, where t_1, t_2 are binary trees.
  - Reasoning: By the IH t_1 has an odd number of nodes. Similarly, so does t_2. Since the number of nodes of $\mathit{node}(t_1, t_2)$ is 1 plus the sum of the nodes of t_1 and t_2, we conclude.

Another Example

- Prove

$$\forall t \in BTree(\mathbb{N}).P(t)$$

$P(t) =$ "t and $incTree(t)$ have the same number of (non-leaf) nodes"

- Recall:

$$\begin{array}{lcl} incTree :: Tree(\mathbb{N}) \to Tree(\mathbb{N}) & & \\ incTree(leaf(n)) & = & leaf(n+1) \\ incTree(node(l, r)) & = & node(incTree(l), incTree(r)) \end{array}$$

- Resort to Structural Induction:
  1. Prove P is true on simple structures (base rules).
  2. Prove that, if P is true on the substructures of t (IH), then it is true on t itself (inductive rules).

Example of Proof using Structural Induction (cont.)

$$\frac{n \in \mathbb{N}}{\mathit{leaf}(n) \in \mathit{BTree}(\mathbb{N})} \qquad \frac{l \in \mathit{BTree}(\mathbb{N}) \quad r \in \mathit{BTree}(\mathbb{N})}{\mathit{node}(l, r) \in \mathit{BTree}(\mathbb{N})}$$

$$\forall t \in \mathit{BTree}(\mathbb{N}).P(t)$$

where $P(t)$ is "t and $\mathit{incTree}(t)$ have the same number of nodes"

- Base case:
  - $t = \mathit{leaf}(i)$, where i is a number.
  - Reasoning: Then $\mathit{incTree}(\mathit{leaf}(i)) = \mathit{leaf}(i + 1)$ and clearly both $\mathit{leaf}(i)$ and $\mathit{leaf}(i + 1)$ have 0 nodes.

Example of Proof using Structural Induction (cont.)

$$\frac{n \in \mathbb{N}}{leaf(n) \in BTree(\mathbb{N})} \qquad \frac{l \in BTree(\mathbb{N}) \quad r \in BTree(\mathbb{N})}{node(l, r) \in BTree(\mathbb{N})}$$

$$\forall t \in BTree(\mathbb{N}).P(t)$$

where $P(t)$ is "t and $incTree(t)$ have the same number of nodes"

- Inductive case:
  - $t = node(t_1, t_2)$, where t_1, t_2 are binary trees.
  - Reasoning: By the IH both t_1 and $incTree(t_1)$ have the same number of nodes. Similarly, both t_2 and $incTree(t_2)$ have the same number of nodes. Therefore, since

    $$incTree(node(t_1, t_2)) \quad = \quad node(incTree(t_1), incTree(t_2))$$

    we may conclude.

Summary

- Inductive Sets: technique for defining sets
- Structural Recursion: technique for defining functions over inductive sets
- Structural Induction: technique for proving properties of inductive sets